HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Decubate
Date: December 06th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for Decubate
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	Liquidity Farming, LP Staking; NFT Staking
Platform	EVM
Network	Ethereum, BSC
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://www.decubate.com/
Timeline	15.09.2022 - 06.12.2022
Changelog	27.09.2022 - Initial Review 20.10.2022 - Second Review 16.11.2022 - Third Review 06.12.2022 - Fourth Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Decubate (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 smart contracts in the repository:

Initial review scope

Repository:
https://github.com/Decubate-com/ats-contracts/releases/tag/v1.16.0
https://github.com/Decubate-com/smart-contracts/blob/liquidity-locker
Commit:
5e9596e02a191d9dc918b52fcdcd0ef4b85bdbe9
0710d188a434ecfc518966da0b1f617dabb5b440
Documentation:
Functional requirements

Integration and Unit Tests: Yes
Deployed Contracts Addresses:-
Contracts:

```
File: ./ats-contracts/contracts/DCBNFTStaking.sol
SHA3: 4b821e8be3041ba69092d020d45b13873f175b6c8aa783661e4ac99f96f8d386

File: ./ats-contracts/contracts/DCBVault.sol
SHA3: 338166bf50b85058a397d873cdb463020f31b51bd48fb1ba15468dca9b9f6195

File: ./ats-contracts/contracts/DecubateMasterChef.sol
SHA3: 1dfbb149f9076348ff23a13a580cce96d4799209604ce77de265c99e16ff5488

File: ./ats-contracts/contracts/interfaces/IDecubateMasterChef.sol
SHA3: f8dbc6f7516e1c30aaed9a861c146b06554d9a11b4a8b3d0999be467cb1d17cf

File: ./contracts/libraries/InterestHelper.sol
SHA3: fe362c644907314c2ab76f59abf2e65118709a004995e6080ff3e47e932f1b07

File: ./smart-contracts/contracts/DCBLiquidityLocker.sol
SHA3: 7df6974c6af0499dededfa60e00eb402fd10d714e6e414670f039c38db95f7e1
```

Second review scope

Repository:
https://github.com/Decubate-com/ats-contracts
Commit:
e9c27417fd3976d9fbbf48c0605aa8caa74694a4
Documentation:
Functional requirements

Integration and Unit Tests: Yes
Deployed Contracts Addresses:-
Contracts:

```
File: ./contracts/DCBLiquidityLocker.sol
SHA3: 12aa5e948f217f5de0b836b6d45020036ada5eee76e7424a5a53a635613d8bd0
```

File: ./contracts/DCBNFTStaking.sol
SHA3: 8f87d35db0451536e87cb34f93ecf13855f737ebdfa7aebd89bc703edaa3d09b

File: ./contracts/DCBVault.sol
SHA3: eac0b7cffa070b270f1a6bda9d2877fa8f986588055963aed0cf71b0faff52d3

File: ./contracts/DecubateMasterChef.sol
SHA3: b5398920dd63834887ea8fe35f99167f05fd5663f5147b7fb937a91e0cc18be0

File: ./contracts/interfaces/IDecubateMasterChef.sol
SHA3: a897b5b0c43c9e00c80e1af2965c22c06fe8359d6e152c1c677513755cc4cfba

File: ./contracts/interfaces/IDecubateStaking.sol
SHA3: 9045e9721762ae8ea5c6a067fb43914734899d73f9ae4d1091b0abeae66a0f88

File: ./contracts/libraries/InterestHelper.sol
SHA3: fe362c644907314c2ab76f59abf2e65118709a004995e6080ff3e47e932f1b07

Third review scope

Repository:
https://github.com/Decubate-com/ats-contracts
Commit:
ccc17a87b1816a32fed634046a139b68a2b52dcb
Documentation:
Functional requirements

Integration and Unit Tests: Yes
Deployed Contracts Addresses:-
Contracts:
File: ./contracts/DCBLiquidityLocker.sol
SHA3: b4241a222cf24feb9b30c4a9cb01bbdf10120914804335db84ef390ad041c17d

File: ./contracts/DCBNFTStaking.sol
SHA3: 48e8e6f87d1d1d53f7d7c7ccd0aec1ba46373cda7d867f4173d61400344ebc72

File: ./contracts/DCBVault.sol
SHA3: b5fdaf6ebfeae4be655243f9febd0290c9df26e3d2ad20c4885a691ed449f6de

File: ./contracts/DecubateMasterChef.sol
SHA3: dd57258c3af489f3e9f76467f99fe6e352a023643310a4f2e4836bbc5611b409

File: ./contracts/interfaces/IDecubateMasterChef.sol
SHA3: decb32db9dfa6a6c84393539c7c8acb33fce9d7de32d4f437ee82d893d96cf98

File: ./contracts/libraries/InterestHelper.sol
SHA3: fe362c644907314c2ab76f59abf2e65118709a004995e6080ff3e47e932f1b07

Fourth review scope

Repository:
https://github.com/Decubate-com/ats-contracts
Commit:
fddd1a87638d6d409c96c8051dd409c06fdc20ad
Documentation:
Functional requirements

Integration and Unit Tests: Yes
Deployed Contracts Addresses:-
Contracts:
File: ./contracts/DCBLiquidityLocker.sol
SHA3: 0de50ca3cb7c1dcaad3ecae5006e060804a21665ed5e5ebed487c146f82079a8

```
File: ./contracts/DCBNFTStaking.sol
SHA3: a257c136d76630e1ecaa8a85f342d12a4eaa4870ebf0e7a61204ea2b67a54e18

File: ./contracts/DCBVault.sol
SHA3: 9325a01086d562b9995d163c6e4dd71023f79e97019182b16b8a9bab90b770ec

File: ./contracts/DecubateMasterChef.sol
SHA3: f8d8f8f070a410d5ec45b55de6b6b1ac456d8749ea47ff036c626afa3f995665

File: ./contracts/interfaces/IDecubateMasterChef.sol
SHA3: 1f62cbbc2dc2f9cb8ed988e208440c6e998ace5d664eb873909d6c2b054d4c8c

File: ./contracts/interfaces/IStaking.sol
SHA3: b90d8595726699721b6b74ae9658ff26686d140a494de791be77a3f1abf6bb62
```

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.

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 **3** out of **10**.

- Partial functional requirements are provided.
- Technical description of DCBNFTStaking and DCBLiquidityLocker was not found in the provided documentation.
- Function parameter explanations in NatSpec format were partially missed.
- Weakly documented README file.

Code quality

The total Code Quality score is **9** out of **10**.

- Code mostly follows the official style guide.
- Testing setup requires a global truffle coverage install pegged to version v5.5.18

Test coverage

Test coverage of the project is **78.5%**.

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is missed.
- Interactions by several users are not tested thoroughly.

Security score

As a result of the audit, the code contains **no issues**. The security score is **10** 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: **8.3**.

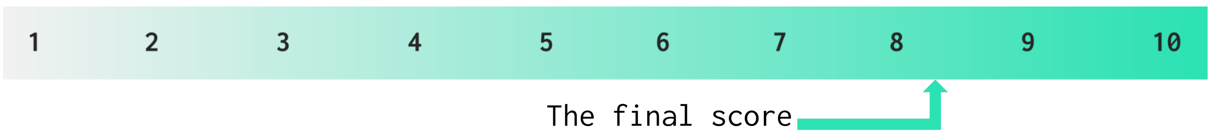

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
21 September 2022	6	5	10	0
18 October 2022	2	0	2	0

15 November 2022	0	0	1	1
05 December 2022	0	0	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.	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.	Passed
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.	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.	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.	Not Relevant
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.	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.	Not Relevant
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.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

Decubate - is a mixed-purpose project which includes Staking Liquidity Farming and NFT Staking. It has the following contracts.
The files in the scope:

- DecubateMasterChef - handles the single asset staking. It uses the DCBVault contract to automatically compound user rewards.
- DCBVault - a contract that is responsible for the auto compounding of rewards. Users deposit their token to the DCBVault contract, and the DCBVault contract then deposits to MasterChef.
- DCBNFTStaking - a contract for staking Decubate NFTs and getting rewards accordingly.
- DCBLiquidityLocker - a contract to add liquidity to DCB pools and lock the earned lp tokens.

Privileged roles

- Owner of the DecubateMasterChef can:
  - update fee percent
  - update fee receiver
  - update compounder
  - add a new period to the pool
  - update pools’ end date, hard cap and max transferrable amount information
- Owner of the DCBVaults can:
  - update fee percent
  - pause and unpause the contract
- Owner of DCBNFTStaking can:
  - add a new period to the pool
  - update pools’ end date and hard cap information
  - update pools’ NFT information
- Owner of DCBLiquidityLocker can:
  - add a new period to the pool
  - update pools’ end date information
- Owner of DecubateStaking contract can:
  - update pools’ end date and hard cap information
  - update fee percent and time gap
  - add a new period the pool
  - set the max transfer input and max transfer reward

Risks

- DCBVault functions limited by the *notContract* modifier do not work with multisig wallets and similar constructs.
- If the number of NFTs is big enough, some transactions, such as claim function, can fail due to exceeding Gas.

Findings

■■■■ Critical

1. **Compilation Issues**

   Because too many parameters are provided in the *add* function, stack is too deep, and the contracts are not able to be compiled.

   Paths: ./contracts/DCBLiquidityLocker.sol: add()

   ./contracts/DCBNFTStaking.sol: add()

   ./contracts/DecubateMasterChef.sol: add()

   Recommendation: Optimize the number of parameters.

   Status: Fixed (Revised commit: fddd1a87638d6d409c96c8051dd409c06fdc20ad)

■■■ High

1. **Checks-Effects-Interactions Pattern Violation**

   There are state variable changes after token transfer in unStake and _stake functions of DecubateMasterChef, harvest and _earn functions of DCBVault, _claim function of DCBLiquidityLocker and safeTransferFrom in unstake and _safeTOKENTransfer in _claim functions of DCBNFTStaking which violates Checks-Effects-Interactions pattern.

   Paths: ./contracts/DecubateMasterChef.sol: unstake(), _stake()

   ./contracts/DCBVault.sol: harvest(), _earn()

   ./contracts/DCBLiquidityLocker.sol: _claim()

   ./contracts/DCBNFTStaking.sol: unStake()

   Recommendation: Implement Check-Effect-Interactions pattern.

   Status: Fixed (Revised commit: e9c2741)

2. **Possible Denial of Service Vulnerability**

   ownsCorrectNFT, and _walletOfOwner functions of DCBLiquidityLocker and ownsCorrectNFT and walletOfOwner of DecubateMasterChef iterates over the NFT's, getPools function of DCBNFTStaking iterates over pools, walletOfOwner of DCBNFTStaking iterates over NFT's owned by address. However, in case of a user has a lot of NFTs or the amount of pool increases excessively, those iterations may fail because of out of Gas.

   Paths: ./contracts/DCBLiquidityLocker: _ownsCorrectNFT(), _walletOfOwner()

./contracts/DecubateMasterChef.sol: ownsCorrectNFT(), walletOfOwner()

./contracts/DCBNFTStaking.sol: getPools()

Recommendation: Provide start and end index for the iteration.

Status: Mitigated (Customer stated that they would show a proper warning on the frontend regarding the exceeding Gas issue if the number of own NFTs is much enough.) (Revised commit: e9c2741)

3. **Possible Denial of Service Vulnerability**

stake and unStake functions of DCBLiquidityLocker iterate over the provided id list; however, there is no limit for the provided id list length. Providing numerous ids may cause fail of transaction because of Gas.

Path: ./contracts/DCBNFTStaking.sol: stake(), unStake()

Recommendation: Check the length of the provided id list.

Status: Mitigated (Customer stated that they would show a proper warning on the frontend regarding the exceeding Gas issue if the number of own NFTs is much enough.) (Revised commit: e9c2741)

4. **Highly Permissive Owner Access**

The owner can withdraw the staking tokens that belong to the users.

Path: ./contracts/DecubateMasterChef.sol: transferToken()

Recommendation: Do not allow withdrawing staking tokens.

Status: Fixed (Revised commit: e9c2741)

5. **Possible Hard Cap Overload**

While reinvesting the staking rewards, pool hard cap can be overloaded since the cap is not checked in *reinvest* function.

Path: ./ats-contracts/contracts/DecubateMasterChef.sol: reinvest()

Recommendation: Check the hard cap for both reinvesting and staking.

Status: Mitigated (Customer stated that the compounder contract is permitted to keep compounding the rewards even if the pool is full. Compounder can bypass hardCap, but users cannot.)(Revised commit: e9c2741)

6. **Insufficient Balance**

During the distribution of the rewards, the total staked balance is not checked. When contract balance - total staked balance is not enough for the requested reward amount, rewards are sent from the staked tokens balance.

This may lead users to lose their original staked tokens due to insufficient balance.

Path: ./contracts/DecubateMasterChef.sol: handleNFTMultiplier(), _claim()

Recommendation: Check if the *total contract balance - staked balance > reward amount* before distributing the rewards.

Status: Fixed (Revised commit: e9c2741)

7. **Highly Permissive Owner Access**

   The owner can withdraw the deposited funds that belong to the users.

   Path: ./contracts/DCBVault.sol: transferToken()

   Recommendation: Do not allow to withdraw deposited pool tokens.

   Status: Fixed (Revised commit: e9c2741)

8. **Highly Permissive Owner Access**

   The owner can withdraw staked NFTs that are belonged to the users.

   Path: ./contracts/DCBNFTStaking.sol: transferStuckNFT()

   Recommendation: Do not allow to withdraw staked NFTs.

   Status: Fixed (Revised commit: e9c2741)

9. **Highly Permissive Owner Access**

   The owner can withdraw the deposited liquidity tokens of users.

   Path: ./contracts/DCBLiquidityLocker.sol: transferToken()

   Recommendation: Do not allow to withdraw liquidity tokens that belong to users.

   Status: Fixed (Revised commit: e9c2741)

10. **Highly Permissive Owner Access**

    Pool info such as APY, lock period days, is withdraw locked status can be changed by the owner even if there are funds of users who have already deposited on those pools.

    This may lead users to lose staked assets when the input address is changed and unstake&withdraw is no longer callable with the old input(asset) address.

    Paths:

    ./contracts/DCBNFTStaking.sol: set(), setNFTInfo()

    ./contracts/DecubateMasterChef.sol: set(), setNFT()

    ./contracts/DCBLiquidityLocker.sol: set(), setNFT()

    Recommendation: Do not change promised rates/rewards when there are users who have already deposited on pools.

Status: Fixed (Revised commit: ccc17a87b1816a32fed634046a139b68a2b52dcb)

11. **Insufficient Balance**

Where there is no sufficient balance for the rewards, unStake function reverts due to making a call to *_claim* function.

Therefore, it is impossible to withdraw the original funds when there is not enough reward balance.

Path:

./contracts/DecubateMasterChef.sol: unStake(),
./contracts/DCBNFTStaking.sol: unStake(),
./contracts/DecubateStaking.sol: unStake()

Recommendation: Build another function to withdraw the funds without rewards.

Status: Fixed (Revised commit: fddd1a87638d6d409c96c8051dd409c06fdc20ad)

■■ Medium

1. **Tautology**

On line 392, lastPayout and depositTime are compared, and one is chosen. However, during every depositing operation, block.timestamp is assigned to both lastPayout and depositTime. Therefore, lastPayout can not be smaller than the depositTime.

Path: ./contracts/DCBNFTStaking.sol : payout()

Recommendation: Remove the redundant comparison and variable.

Status: Fixed (Revised commit: e9c2741)

2. **Unfinished Code**

Two functions located on lines 340-361 are commented. This reduces the code quality and may lead to confusion for users.

Path: ./contracts/DCBNFTStaking.sol

Recommendation: Delete the commented code parts or implement the unfinalized code.

Status: Fixed (Revised commit: e9c2741)

■ Low

1. **Floating Pragma**

The project uses floating pragmas ^0.8.10, ^0.8.15.

Paths: all

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

Status: Fixed (Revised commit: e9c2741)

2. **Redundant Use of SafeMath**

Since Solidity v0.8.0, the overflow/underflow check is implemented via ABIEncoderV2 on the language level - it adds the validation to the bytecode during compilation.

There is no need to use the SafeMath library.

Paths: all

Recommendation: Remove the SafeMath library.

Status: Fixed (Revised commit: ccc17a87b1816a32fed634046a139b68a2b52dcb)

3. **Unchecked Return Value**

The return value of a message call is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behavior in the subsequent program logic.

Return value of safeTOKENTransfer in handleNFTMultiplier, unStake and _claim functions of DecubateMasterChef are not checked.

Return value of approve in deposit and unStake in withdraw functions of DCBVaults are not checked.

Return value of transfer in _earn and _safeTOKENTransfer, transferFrom in _addLiquidity and _addLiquidityETH of DCBLiquidityLocker are not checked.

Paths: ./contracts/DecubateMasterChef: handleNFTMultiplier()

./contracts/DCBVault.sol: deposit(), withdraw()

./contract/DCBLiquidityLocker.sol: _earn(), _addLiquidity(), _safeTOKENTransfer(), _addLiquidityETH()

Recommendation: Implement a check of the returning value.

Status: Fixed (Revised commit: e9c2741)

4. **Variable Written Twice**

user.common.lastPayout variable is already updated while calling the _claim function. Therefore, re-updating it on line 180 is redundant.

Path: ./contracts/DCBNFTStaking.sol : stake()

Recommendation: Write a variable once.

Status: Fixed (Revised commit: e9c2741)

5. **Redundant Variable**

totalClaimed and *totalWithdrawn* variables keep the same value. Redundant variable declarations decrease the Gas consumption and increase the code complexity.

Path: ./contracts/DecubateMasterChef.sol

Recommendation: Remove one of the variables.

Status: Fixed (Revised commit: e9c2741)

6. **Style Guide Violation**

   The provided projects should follow the official guidelines.

   Paths: all

   Recommendation: Follow the official Solidity guidelines: https://docs.soliditylang.org/en/v0.8.13/style-guide.html

   Status: Fixed (Revised commit: ccc17a87b1816a32fed634046a139b68a2b52dcb)

7. **Lost Index**

   add function does not return the new index of the multiplier in *multipliers* array when a pool is created. Therefore, the owner will not be able to set it later since the setting requires an index for *multipliers* array.

   Path: ./contracts/DCBNFTStaking.sol : add()

   Recommendation: Return the index of the new element or emit an event for it.

   Status: Fixed (Revised commit: e9c2741)

8. **Lost Index**

   add function does not return the new index of pool in *poolExt* array. Therefore, the owner will not be able to set it later since the setting requires an index for *poolExt* array.

   Path: ./contracts/DCBNFTStaking.sol : add()

   Recommendation: Return the index of the new element or emit an event for it.

   Status: Fixed (Revised commit: e9c2741)

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.