HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Marhaba DeFi
Date: August 22th, 2022

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 Marhaba DeFi
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Staking
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://www.mrhb.network/
Timeline	12.07.2022 - 22.08.2022
Changelog	22.07.2022 - Initial Review 04.08.2022 - Second Review 22.08.2022 - Third Review

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by Marhaba DeFi (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/Marhaba-DeFi/vote_escrow
Commit:
6b7b74bd00c6b6515268d14a056a58e9c5b13a94
Technical Documentation:
Type: Functional requirements
Integration and Unit Tests: Yes
Contracts:
File: ./src/interfaces/ILocker.sol
SHA3: bd77d8ac670d9e960b662f959e0c73481db1b0a21203d1057fca7d93930c74d1

File: ./src/interfaces/IRewardDistribution.sol
SHA3: 71315b545d4b0448efaae5290228f80e2f4a797564376253b90188a180b566a5

File: ./src/interfaces/IVoter.sol
SHA3: 26343a833bdd34af4d7bdd78cc5d291fee5b75641c7c695991744a828e3276bf

File: ./src/Locker.sol
SHA3: 02bddaa9775b20da7247c276b79f254094ea156bf2a0c74e2beeb8f6ae8ad77c

File: ./src/MockERC20.sol
SHA3: 7902e3d0f35b2107a9286161a210ace9d09321e342f39ca8cd091ac1b879b2ea

File: ./src/RewardDistribution.sol
SHA3: 878bc2634b58d507d7365683040b7fea44687efd80c8b83d507824c9a0eb55d2

File: ./src/Voter.sol
SHA3: 4d75297185ec6e78e3ba1bff2eec3780d1525bbc221b2aedfc3600833bc5806b

File: ./test/EdgeCases.t.sol
SHA3: 83d415c58d91c2cce5c6e5dc8758431c76475398992f7793855432dcda3d3927

File: ./test/Fixture.sol
SHA3: e5f09e49c90d1c798575208932ad2b84c583e6f5ab333fd1de70af10a5acce8b

File: ./test/Locker.t.sol
SHA3: fa003f6282151dad1ec85153b7354b6c4c014a09a0bc330f05b663a4fc9c311e

File: ./test/RewardDistribution.t.sol
SHA3: 5db7c6cbd3b3460b575fea74e1f3c5a9a5afd038616b28266972e9418c89e793

File: ./test/TestScenario.t.sol
SHA3: 212c18126b5d4a00ed100adb3830e55dd27c7aef77da5cc3e7e2a0dd07a7b3ac

Second review scope

Repository:
https://github.com/Marhaba-DeFi/vote_escrow
Commit:
c77cbf0143e874ac0d7a8161413cdff731bd9545
Technical Documentation:
Type: Functional requirements
Integration and Unit Tests: Yes
Contracts:

File: ./src/interfaces/ILocker.sol
SHA3: 72d8c667c208a64e71bbec9a65d137b272603dce85a19e6d5056d42c3aa25681

File: ./src/interfaces/IRewardDistribution.sol
SHA3: 9772d36e65ad85e192c5eea5d60413649c3bf42eccc728c75ae8a0c32628aff9

File: ./src/interfaces/IVoter.sol
SHA3: 6760d1897773b63c85577a682c7f77ece24664a3ebf6d4f86223cf5dd24dc14f

File: ./src/Locker.sol
SHA3: d3af4e6ead5cc73de95e270f4fdec38b1922c9cc1a48c0b9de3384324d8da0a0

File: ./src/MockERC20.sol
SHA3: 7902e3d0f35b2107a9286161a210ace9d09321e342f39ca8cd091ac1b879b2ea

File: ./src/RewardDistribution.sol
SHA3: 1200091abd360c7f3b81ac728a8aba591777fed6365e239f3f0fced5109abfbe

File: ./src/Voter.sol
SHA3: 4883db1d0c36a8786064461bdc9bedca04670033a310961806f32710491415e8

File: ./test/EdgeCases.t.sol
SHA3: 1f7c653614f7f93e900889eff81cc7f84bb5b64082c3abb45c5bbb486f0bc0ae

File: ./test/Fixture.sol
SHA3: 857b21a9bce36d50328e4ce319f1ef6815a940dbc644f508192f4b6db729baa0

File: ./test/Locker.t.sol
SHA3: 74e940bb4657ea66f156468752a66b0e31afe584ce19974fc957cab30048802f

File: ./test/RewardDistribution.t.sol
SHA3: c1140433afaa19e50d947dac36d82bfddee47a3548d151941db06bf8183cb75b

File: ./test/TestScenario.t.sol
SHA3: a1dcef2cd3107aa9853ef4f191dd5af7ba28618210d82b11802526d25a9cedfa

File: ./test/Voter.t.sol
SHA3: 6a68b9ae065c2ea624e875463da7a0c3610c53be53691f7cc6dd568b3cbf6d53

Third review scope

Repository:
https://github.com/Marhaba-DeFi/vote_escrow
Commit:
25a0f3e7418583095b25e89770eb7a6a196174c8
Technical Documentation:
Type: Functional requirements
Integration and Unit Tests: Yes
Contracts:

File: ./src/interfaces/ILocker.sol

```
SHA3: 72d8c667c208a64e71bbec9a65d137b272603dce85a19e6d5056d42c3aa25681

File: ./src/interfaces/IRewardDistribution.sol
SHA3: 9772d36e65ad85e192c5eea5d60413649c3bf42eccc728c75ae8a0c32628aff9

File: ./src/interfaces/IVoter.sol
SHA3: 6760d1897773b63c85577a682c7f77ece24664a3ebf6d4f86223cf5dd24dc14f

File: ./src/Locker.sol
SHA3: 1a57c8975b66bf0e574740dba0500c9217d075ee93263794bbd6e9bc6a254e89

File: ./src/MockERC20.sol
SHA3: 7902e3d0f35b2107a9286161a210ace9d09321e342f39ca8cd091ac1b879b2ea

File: ./src/RewardDistribution.sol
SHA3: ef6d267b693397a6bea8161be5f1ad3040bb9e401483bb37169d0dc0f0748fe2

File: ./src/Voter.sol
SHA3: 801b9969ea245c7c8fce52d9fab724617503acaae47a42f7a5de2ba2f2e23751

File: ./test/EdgeCases.t.sol
SHA3: 1f7c653614f7f93e900889eff81cc7f84bb5b64082c3abb45c5bbb486f0bc0ae

File: ./test/Fixture.sol
SHA3: 857b21a9bce36d50328e4ce319f1ef6815a940dbc644f508192f4b6db729baa0

File: ./test/Locker.t.sol
SHA3: 74e940bb4657ea66f156468752a66b0e31afe584ce19974fc957cab30048802f

File: ./test/RewardDistribution.t.sol
SHA3: c1140433afaa19e50d947dac36d82bfddee47a3548d151941db06bf8183cb75b

File: ./test/TestScenario.t.sol
SHA3: a1dcef2cd3107aa9853ef4f191dd5af7ba28618210d82b11802526d25a9cedfa

File: ./test/Voter.t.sol
SHA3: 6a68b9ae065c2ea624e875463da7a0c3610c53be53691f7cc6dd568b3cbf6d53
```

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

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 methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**. Functional requirements are comprehensive. A technical description is provided as the detailed comments in the code.

Code quality

The total CodeQuality score is **10** out of **10**. Most of the code follows official language style guides. Tests were provided.

Architecture quality

The architecture quality score is **8** out of **10**. The development environment was provided, the description of how to compile, build, test the code was provided, but the description of how to deploy the code was not provided. The code contains circular dependencies.

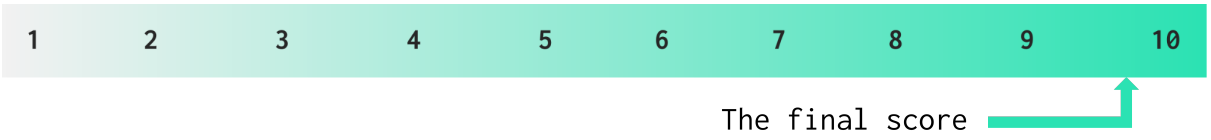
Security score

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: **9.8**.



Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are 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.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	SWC-115	tx.origin should not be used for	Passed

through tx.origin		authorization.	
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 identifier 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.	Not Relevant
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	Passed

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
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.	Failed
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

MIRO is a staking system with the following contracts:

- *Locker* — is a contract for MRHB tokens (token address is set during the contract deployment) locking.
  Users get the vMRHB tokens for the condition in the amount of locked MRHB tokens multiplied by the coefficient specified for the allowed locking period:
    - 13 weeks: Coefficient of 0.1
    - 26 weeks: Coefficient of 0.21
    - 52 weeks: Coefficient of 0.45
    - 104 weeks: Coefficient of 1

  vMRHB tokens are totally added to the balance at once and decrease weekly to 0 until the exceeding of each locking. Weekly vMRHB amount: *total initial vMRHB balance * (1 - current locking week / the staking period in weeks)*.

  Users can extend locks. Added lock period should be included in the allowed locking period list (13, 26, 52, 104 weeks). The extended lock should not exceed the allowed locking period. When extending the lock, the newly added amount multiplied by the appropriate coefficient is added to the current user`s vMRHB balance, the lock end week is changed, and the user`s vMRHB balances are recalculated for the current and all the next locking weeks.

  Users can increase the locking amounts. The lock should not be exceeded. The vMRHB for the current week is recalculated by the sum of already existing vMRHB tokens amount and vMRHB for the newly added MRHB tokens: *the added amount * user`s current vMRHB / user`s staked MRHB tokens (without newly added amount)*. The user`s vMRHB balances are recalculated for the current and all the next locking weeks.

  Each user can have one locking. There is a limit of maximal locking weeks. It is defined during the contract deployment.

  There are three tiers for lockings: gold, silver, and bronze. They are determined depending on the number of tokens locked by users and the locking periods: for getting gold or silver tier, the locking period should be equal to or greater than 52 weeks, the staked amounts must fall within the specified ranges defined by the owner. All the other users get the bronze tiers. The tiers are defined off-chain.

  As the locking expires, it is possible to withdraw locked funds. The withdrawal of tokens to the locker address can be triggered by any user.

  The contract contains migrations functionality: if the contract moves to a new address, the owner may allow users to withdraw their stakes

without waiting for their lock to expire. (If the lock expires, users should withdraw their funds in the usual way, as the migration funds withdrawal reverts for the expired locks).

- *Voter* – is a contract for users voting using the vMRHB tokens. 1 vMRHB = 1 vote. The owner opens portals (up to 8) and creates the proposal. Users can vote on different proposals using different amounts of vMRHB tokens.
  When user votes, the claimable reward is calculated for him/her: *the total deposited rewards to RewardDistribution contract for the current week / the total amount of vMRHB tokens that exist this week * the vMRHB amount of the user*.

  If a user does not vote, his/her rewards tokens will be shared between all users the next week.
- *RewardDistribution* – is a contract for the rewards distributions for the MRHB locking. Users with "Rewarders" role deposit tokens that will be paid as rewards. According to the rewards calculations in *Voter* contract, users can claim their rewards at any time. Users can set the receiver address for their rewards. The claiming of each user`s rewards (to the receiver address) can be triggered by any user.
  The rewards are paid in the MRHB tokens.
- *MockERC20* – is a ERC-20 token contract for testing purposes. It has the following attributes:
  - name: "Mock"
  - symbol: "MCK"

  The total supply is unlimited.
- *ILocker* – is an interface for the *Locker* contract, used in *Voter* and *RewardDistribution* contracts.
- *IRewardDistribution* – is an interface for the *RewardDistribution* contract, used in *Voter* contract.
- *IVoter* – is an interface for the *Voter* contract, used in the RewardDistribution contract.
- *EdgeCasesTests, Fixture, LockerTests, RewardDistributionTests, TestScenario* – are the contracts for tests.

Privileged roles

- The owner of the Locker contract can withdraw ERC-20 tokens from the contract (except MRHB tokens), set the contract migration status, which allows users to withdraw their stakes without waiting for locks to expire.
- The owner of the *Voter* contract can set and update the *RewardDistribution* address and open voting portals and close them, and withdraw ERC-20 tokens from the contract (except MRHB tokens).

- The RewardDistribution of the *Voter* contract can reset users' claimable rewards amount (is used in the *RewardDistribution* contract, when users withdraw their rewards).
- The owner of the *RewardDistribution* contract can add and remove Rewarders, and withdraw ERC-20 tokens from the contract (except MRHB tokens).
- The “Rewarders” of the *RewardDistribution* contract can deposit the tokens for rewards.
- The “governance” of the *MockERC20* contract can change governance address, set “minters”, mint, and burn tokens.
- The “minters” of the *MockERC20* contract can mint tokens.

Risks

- Rewards payment can not be guaranteed as rewards are transferred by the “Rewarders” of *RewardDistribution* contract.
- Tiers and ranges for tiers are set off-chain (out of scope), and cannot be guaranteed. Granting special utils for the tiers is out of scope and cannot be verified (All the users have the same access to the voting within the given project).
- If depositing rewards to *RewardDistribution* contract on the first day of a week, this will lead to incorrect rewards, as the users may lock tokens on the same day, and the rewards for them will be taken into account for the same week.
- Voting portals are opened and closed by the owner, and can not be guaranteed.
- *MockERC20* contract allows the user with the “governance” role to burn any tokens. The total supply of tokens is unlimited; users with “governance” and “minter” roles can mint tokens.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

1. **Denial of Service vulnerability**

   The rewards are calculated in a loop that runs over an array with all the weekly rewards tokens.

   If the array with tokens is large enough to make the Gas required for executing the for loop exceed the block Gas limit, the *vote* function will be inoperable.

   Path: ./src/Voter.sol

   Contract: Voter

   Function: vote

   Recommendation: Limit the number of iterations.

   Status: Fixed (Revised commit: c77cbf0143e874ac0d7a8161413cdff731bd9545)

2. **Data consistency**

   Actual tiers of users are not updated during the tiers threshold update.

   This can lead to an inconsistent contract state.

   Path: ./src/Locker.sol

   Contract: Locker

   Function: setTiers

   Recommendation: Calculate user tier dynamically and do not store this value.

   Status: Fixed (Revised commit: c77cbf0143e874ac0d7a8161413cdff731bd9545)

3. **Requirements violation**

   According to the comments in code, the rewards should be claimed from the *_user* address balance in the *claim* function. However, the rewards are claimed from the *msg.sender* address.

   Therefore, the rewards will be withdrawn from the function caller address instead of the *_user* address.

   Path: ./src/Locker.sol

   Contract: RewardDistribution

Function: claim

Recommendation: Implement the code according to the requirements.

Status: Fixed (Revised commit: 25a0f3e7418583095b25e89770eb7a6a196174c8)

Medium

1. **Checks-Effects-Interaction pattern violation**

   The state variables are updated after the external calls.

   This can lead to reentrancies, race conditions, or denial of service vulnerabilities.

   Paths: ./src/Locker.sol, ./src/Voter.sol, ./src/RewardDistribution.sol

   Contracts: Locker, Voter, RewardDistribution

   Functions: Locker.increaseAmount, Voter.vote, RewardDistribution.depositRewards

   Recommendation: Implement functions according to the checks-effects-interactions pattern.

   Status: Mitigated. The Customer comment: “Check effects are not a concern here there is no potential reentrancy”

2. **Missing events emit on changing important values**

   The contract does not emit any events after changing important values

   Paths: ./src/RewardDistribution.sol, ./src/Voter.sol

   Contracts: RewardDistribution, Voter

   Functions: RewardDistribution.setClaimReceiver, RewardDistribution.setRewarder, Voter.setRewardDistributor

   Recommendation: Implement event emits after changing the contract values.

   Status: Fixed (Revised commit: 25a0f3e7418583095b25e89770eb7a6a196174c8)

3. **Unsafe role granting**

   The functionality allows the owner to set the addresses, which are allowed to deposit rewards (*rewarders*), but not to remove them.

   If the defined address is compromised, irrelevant, or wrongly specified, it will not be possible to change it. Incorrect rewards sending leads to the DoS vulnerability and insufficient funds.

   Path: ./src/RewardDistribution.sol

   Contract: RewardDistribution

Functions: setRewarder, depositRewards

Recommendation: Add the functionality to remove addresses from *rewarders*. It is recommended to use only one address for the reward distribution to ensure that tokens are deposited once a week at the correct time.

Status: Fixed (Revised commit: c77cbf0143e874ac0d7a8161413cdff731bd9545)

4. **Redundant input parameter**

_coefficient parameter is redundant. The value can be obtained from the *weeklyCoefficients* mapping.

Path: ./src/Locker.sol

Contract: Locker

Function: lock

Recommendation: Obtain the *_coefficient* value from the *weeklyCoefficients* mapping.

Status: Fixed (Revised commit: c77cbf0143e874ac0d7a8161413cdff731bd9545)

5. **Missing zero value validation**

_weeks parameter is not validated for a non-zero value.

Due to this, division by zero is possible at Line#181

Path: ./src/Locker.sol

Contract: Locker

Function: lock

Recommendation: Verify that *_weeks* parameter is non-zero.

Status: Fixed (Revised commit: c77cbf0143e874ac0d7a8161413cdff731bd9545)

Low

1. **Redundant functionality**

The ** WEEK / WEEK* is redundant in the current time calculation.

File: ./src/Locker.sol

Contract: Locker

Function: getCurrentTime

Recommendation: Remove the redundant code.

Status: Fixed (Revised commit: c77cbf0143e874ac0d7a8161413cdff731bd9545)

2. **Redundant functionality**

   The *received* definition is redundant, as the *_amount* value can be used directly.

   Path: ./src/RewardDistribution.sol

   Contract: RewardDistribution

   Function: depositRewards

   Recommendation: Remove the redundant code.

   Status: Fixed (Revised commit: c77cbf0143e874ac0d7a8161413cdff731bd9545)

3. **Floating pragma**

   The project`s contracts use floating pragma ^0.8.0, ^0.8.13.

   Contracts with unlocked pragmas may be deployed by the latest compiler, which may have higher risks of undiscovered bugs. Contracts should be deployed with the same compiler version they have been tested thoroughly.

   Files: ./src/interfaces/ILocker.sol, ./src/interfaces/IRewardDistribution.sol, ./src/interfaces/IVoter.sol, ./src/Locker.sol, ./src/MockERC20.sol, ./src/RewardDistribution.sol, ./src/Voter.sol, ./test/Contract.t.sol, ./test/EdgeCases.t.sol, ./test/Fixture.sol, ./test/Locker.t.sol, ./test/RewardDistribution.t.sol, ./test/TestScenario.t.sol

   Contracts: ILocker, IRewardDistribution, IVoter, Locker, MockERC20, RewardDistribution, Voter, ContractTest, EdgeCasesTests, Fixture, LockerTests, RewardDistributionTests, TestScenario

   Recommendation: Consider locking the pragma version whenever possible.

   Status: Fixed (Revised commit: c77cbf0143e874ac0d7a8161413cdff731bd9545)

4. **Default visibility usage**

   There are variables in the contract whose visibilities are not defined explicitly.

   This may lead to incorrect access to variables.

   File: ./src/RewardDistribution.sol

   Contract: RewardDistribution

   Paths: seenRewards, rewarders

   Recommendation: Define the variables' visibilities explicitly.

Status: Fixed (Revised commit: c77cbf0143e874ac0d7a8161413cdff731bd9545)

5. **Redundant modifier usage**

   The *withdraw* function uses a nonReentrant modifier, but the function is implemented according to the checks-effects-interactions pattern.

   File: ./src/Locker.sol

   Contract: Locker

   Function: getCurrentTime

   Recommendation: Remove the redundant modifier usage.

   Status: Fixed (Revised commit: c77cbf0143e874ac0d7a8161413cdff731bd9545)

6. **Block values as a proxy for time using**

   The contract uses block.timestamp for time calculations. It is not precise and safe.

   Paths: ./src/Locker.sol, ./src/Voter.sol

   Contracts: Locker, Voter

   Functions: Locker.getWeek, Locker.getCurrentTime, Voter.closePortal, Voter.openPortal

   Recommendation: It is recommended to avoid using block.timestamp. Alternatively, it is safe to use oracles.

   Status: Mitigated. The Customer comment: “We use block timestamp to check differences in days and sometimes weeks. Given the gap timestamp should not be a concern as a miner cannot manipulate timestamp by days and weeks due to structure of blockchain and validations.”

7. **Unused variables**

   Unused variables should be removed from the contracts. Unused variables are allowed in Solidity and do not pose a direct security issue. It is best practice to avoid them as they can cause an increase in computations (and unnecessary Gas consumption) and decrease readability.

   Files: ./src/RewardDistribution.sol, ./src/Voter.sol

   Contracts: RewardDistribution, Voter

   Paths: RewardDistribution.WEEK, RewardDistribution.startTime, Voter.startTime

   Recommendation: Remove unused variables.

   Status: Fixed (Revised commit: 25a0f3e7418583095b25e89770eb7a6a196174c8)

8. Code duplication

Functions contain code duplications that can be easily optimized.

File: ./src/RewardDistribution.sol

Contracts: RewardDistribution, Voter

Paths: claimAll, claim

Recommendation: Move common code to a separate function.

Status: Fixed (Revised commit: c77cbf0143e874ac0d7a8161413cdff731bd9545)

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.