HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: BreederDAO
Date: April 14th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for BreederDAO.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	ERC20 token; Vesting
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.breederdao.io
Timeline	08.04.2022 - 14.04.2022
Changelog	12.04.2022 - Initial Review 14.04.2022 - Revision

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by BreederDAO (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:
Repository:
https://github.com/breederdao/audit-contract-hacken
Commit:
4e266c3a6ccf225e40ad479fdba21c95b0fc952f
7dbb3d56713c632f1c97c6c1d0e32ac95df05876 (revision)
Documentation: Yes
(https://github.com/breederdao/audit-contract-hacken/blob/develop/README.md)
JS tests: Yes
(https://github.com/breederdao/audit-contract-hacken/tree/develop/test)
Contracts:
IBreederDaoTokenLock.sol
IBreederDaoTokenLockManager.sol
IBreederToken.sol
BreederDaoTokenLock.sol
BreederDaoTokenLockManager.sol
BreederDaoTokenLockWallet.sol
BreederToken.sol
MinimalProxyFactory.sol
Ownable.sol
Owned.sol
MathUtils.sol
Pausable.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency ▪ Kill-Switch Mechanism

Executive Summary

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

Documentation quality

The Customer provided good functional and technical requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **10** out of **10**.

Architecture quality

The architecture quality score is **10** out of **10**. The logic is carefully separated by several files.

Security score

As a result of the audit, security engineers found **1** low severity issue. The security score is **10** out of **10**. All found issues are displayed in the "Issues overview" section.

Summary

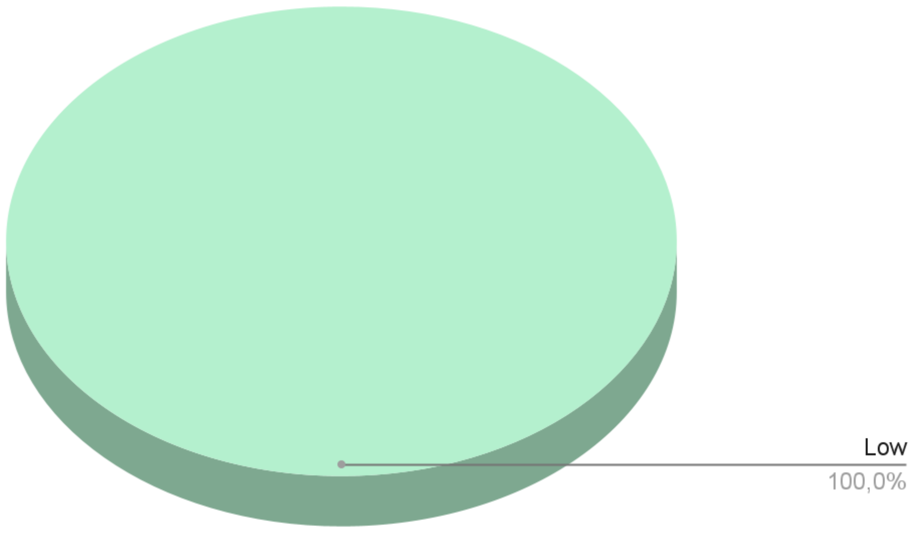
According to the assessment, the Customer's smart contract has the following score: **10**.

1
2
3
4
5
6
7
8
9
10
You are here

Notices

1. After the initial review there were added several new contracts that are not included to the audit scope.
2. The owner can revoke a vesting and take back unvested money, but rewards earned up to that moment would still be available.
3. The owner can change the master version of the vesting contract. The audit only covers the version from the scope.

Graph 1. The distribution of vulnerabilities after the audit.



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

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟨🟨🟨 High

1. Unexpected allowance

Approving and revoking the protocol by the beneficiary may leave some additional allowances. If the admin has removed or added any destination between user approving and revoking, some unexpected allowances or disallowances may appear.

This could lead to the disappearance of vested tokens that are not approved to the protocol.

Contract: BreederDaoTokenLockManager.sol

Functions: addTokenDestination, removeTokenDestination

Recommendation: manage and update all wallet approvals on adding or removing destinations.

Status: Fixed (Revised Commit: 7dbb3d5)

🟩🟩 Medium

1. Corrupted calculation of periods

Constant *MIN_PERIOD* is the minimum amount of vesting periods, but it is also used for calculating the current period number. That is wrong because for calculation should be used adding or subbing just *1*, not a constant linked to the amount of vesting periods.

In such a way, the logic of the functions may be corrupted if *MIN_PERIOD* is changed in the future.

Contract: BreederDaoTokenLock.sol

Functions: currentPeriod, passedPeriods, availableAmountByTimestamp

Recommendation: replace *MIN_PERIOD* with 1 in mentioned functions.

Status: Fixed (Revised Commit: 7dbb3d5)

2. Possible gas exceeding

The contract can exceed the gas limit in long cycles or by returning a big array.

Contract: BreederDaoTokenLockManager.sol

Function: getTokenDestinations

Contract: BreederDaoTokenLockWallet.sol

Functions: approveProtocol, revokeProtocol

Recommendation: use page navigation to avoid unexpected exceeding of Gas; the user should have the ability to separate his action in several transactions if needed.

Status: Fixed (Revised Commit: 7dbb3d5)

Low

1. Floating pragma

The contracts use floating pragma ^0.8.10 and ^0.8.6.

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

Status: Fixed (Revised Commit: 7dbb3d5)

2. Local variable shadowing

Local variable shadowing may lead to unexpected code behavior in future development.

Contract: BreederDaoTokenLock.sol

Functions: availableAmountByTimestamp:(currentPeriod), _initialize:(_owner)

Contract: BreederDaoTokenLockWallet.sol

Function: initialize:(_owner)

Contract: BreederDaoTokenLockManager.sol

Functions: createTokenLockWallet:(_owner)

Recommendation: rename the local variables that shadow other components.

Status: Fixed (Revised Commit: 7dbb3d5)

3. Boolean equality

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

Contract: BreederDaoTokenLock.sol

Function: cancelLock, revoke

Recommendation: remove the equality to the boolean constant.

Status: Fixed (Revised Commit: 7dbb3d5)

4. Unused event

The *TokenDestinationAllowed* event was declared, but never used.

Contract: BreederDaoTokenLockManager.sol

Recommendation: remove the unused event.

Status: New

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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

Technical Disclaimer

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