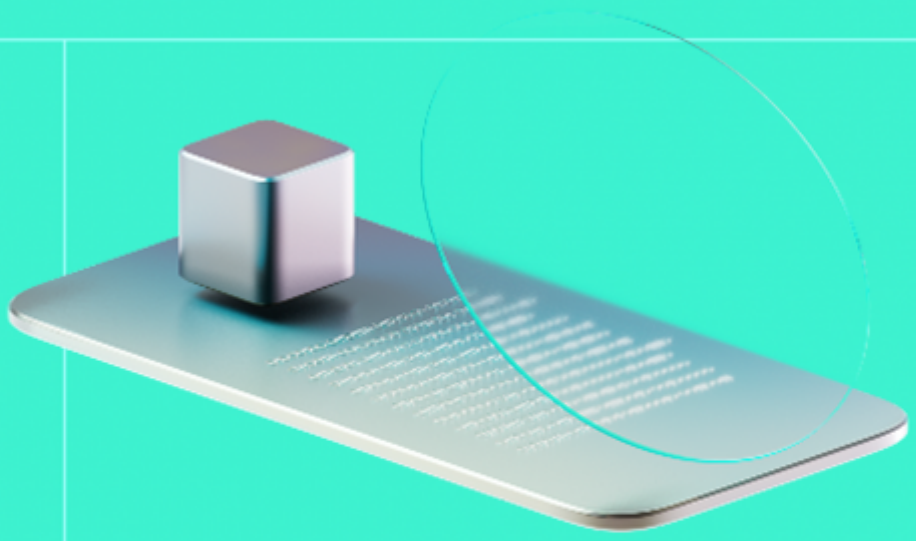
HACKEN

# Smart Contract Code Review And Security Analysis Report

**Customer:** Dogami

**Date:** 23/01/2024



We express our gratitude to the Dogami team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

DOGAMÍ is a staking protocol that allow users to lock an ERC20 Token for either a fixed (limited) or open-ended (unlimited) period and earn rewards in the same ERC20 Token.

**Platform:** EVM

**Language:** Solidity

**Tags:** Staking

**Timeline:** 16/01/2024 - 23/01/2024

**Methodology:** https://hackenio.cc/sc_methodology

## Review Scope

| | |
|---|---|
| **Repository** | https://github.com/dogami-code/Smart-Contracts-EVM |
| **Commit** | 2276c97 |

## Audit Summary

| 10/10 | 10/10 | 97% | 10/10 |
|---|---|---|---|
| Security Score | Code quality score | Test coverage | Documentation quality score |

# Total 9.9/10

The system users should acknowledge all the risks summed up in the risks section of the report

| 4 | 4 | 0 | 0 |
|---|---|---|---|
| Total Findings | Resolved | Accepted | Mitigated |

### Findings by severity

| | |
|---|---|
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 4 |

| Vulnerability | Status |
|---|---|
| F-2024-0493 - Missing return value check for tokens transfers may lead to unexpected behavior | Fixed |
| F-2024-0494 - Missing checks for reward ratio numerator and denominator can lead to incorrect reward values | Fixed |
| F-2024-0501 - Missing the timeUnit variable check can lead to division by zero in reward calculations | Fixed |
| F-2024-0502 - Potential misuse of user funds in reward distribution due to incorrect constructor settings | Fixed |

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

## Document

| | |
|---|---|
| Name | Smart Contract Code Review and Security Analysis Report for Dogami |
| Audited By | David Camps Novi |
| Approved By | Przemyslaw Swiatowiec |
| Website | https://dogami.com |
| Changelog | 18/01/2024 - Preliminary Report; 23/01/2024 - Final Report |

# Table of Contents

## System Overview

DOGAMÍ is a staking protocol that allow users to lock an ERC20 Token for either a fixed (limited) or open-ended (unlimited) period and earn rewards in the same ERC20 Token.

- The `StakingFlex` contract allows for unlimited staking at a fixed rate, which can be later updated by the admin wallet if needed.
- The `StakingLockPeriod` contract allows for staking for a pre-defined period at a fixed rate that cannot be later updated by the admin wallet.

## Privileged roles

Both staking contracts have 2 roles:

- `Admin`: Can pause/unpause the contract if needed for both the `StakingFlex` and `StakingLockPeriod` contracts. In addition, for the `StakingFlex` contract, `admin` can update the time unit over which the reward is calculated. The `admin` can also modify the reward ratio. Can force a user to un-stake their funds.
- `User`: Can stake tokens, collect their rewards, and withdraw parts or the entirety of their stacked tokens.

# Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

## Documentation quality

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

- Functional requirements are provided.
- Technical description is provided.

## Code quality

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

- Best practices are followed .
- The development environment is configured.

## Test coverage

Code coverage of the project is **97%** (branch coverage).

- Main deployment and basic user interactions are covered with tests.

## Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **0** medium, and **4** low severity issues. All issues were fixed, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

## Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.9**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

## Risks

- The reward tokens are sent from the `rewardWallet` address, but the balance of that address is checked off-chain. It may be the case that a user engages with the system when no rewards are available. Another possibility is that no rewards can be sent out to a user when the rewards are claimed, reverting the call that withdraws the tokens and receives the rewards

# Findings

## Vulnerability Details

### F-2024-0493 - Missing return value check for tokens transfers may lead to unexpected behavior - Low

**Description:**

In the safeTransferERC20 functions, both `transfer` and `transferFrom` ERC20 functions are called to transfer tokens. However, the function does not check the `return` values of these calls.

Not all ERC20 tokens are guaranteed to `revert` on failure; some may `return` a boolean value (`false`) instead. If the system interacts with such tokens, a failed transfer would not cause the transaction to `revert`, potentially leading to discrepancies in the contract's state.

Other ERC20 do not return anything at all when calling `transfer` and `transferFrom`. Therefore, every call will be reverted when the `return` value is checked.

The following functions are affected:
`StakingLockPeriod: _safeTransferERC20.`
`StakingFlex: safeTransferERC20.`

**Assets:**

- StackingFlex.sol [https://github.com/dogami-code/Smart-Contracts-EVM/commit/3b063ec71c18aa313cf614977ca20a18daa0c235]
- StackingLockPeriod.sol [https://github.com/dogami-code/Smart-Contracts-EVM/tree/WIP/Stacking]

**Status:** Fixed

#### Classification

**Severity:** Low

**Impact:** 4/5

**Likelihood:** 3/5

#### Recommendations

**Recommendation:**

Check the `return` value of the calls to ERC20 `transfer` and `transferFrom`. Additionally, implement the SafeERC20 library to interact with safely with tokens that do not return anything at all.

**Remediation (revised commit: 2276c97):** The SafeERC20 library was implemented.

## F-2024-0494 - Missing checks for reward ratio numerator and denominator can lead to incorrect reward values - Low

**Description:**

The staking contracts use `rewardRatioNumerator` and `rewardRatioDenominator` in order to set the reward tokens obtained from staking tokens.

Due to the lack of checks that limit their values, the reward ratio could be set to 0% or higher than 100%, leading to unwanted reward values.

The following parameters are affected:
`StakingLockPeriod: constructor → _numerator, _denominator.`
`StakingFlex: constructor, setRewardRatio, _setStakingCondition → _numerator, _denominator.`

**Assets:**

- StackingFlex.sol [https://github.com/dogami-code/Smart-Contracts-EVM/commit/3b063ec71c18aa313cf614977ca20a18daa0c235]
- StackingLockPeriod.sol [https://github.com/dogami-code/Smart-Contracts-EVM/tree/WIP/Stacking]

**Status:** Fixed

### Classification

**Severity:** Low

**Impact:** 4/5

**Likelihood:** 2/5

### Recommendations

**Recommendation:**

Reasonable limits should be added to the variables `_numerator` and `_denominator` to prevent unexpected side effects.

**Remediation (revised commit: 2276c97):** Checks were implemented to avoid setting values to `0` or `numerator` being higher than `denominator`..

## F-2024-0501 - Missing the timeUnit variable check can lead to division by zero in reward calculations - Low

**Description:**

The parameter `_timeUnit` is critical for determining the time unit in reward calculations. It is essential that this parameter is set to a non-zero value to prevent division by zero errors during the computation of rewards.

A check should be implemented in the following cases:
`StakingFlex: constructor, setTimeUnit.`
`StakingLockPeriod: constructor.`

**Assets:**

- StackingFlex.sol [https://github.com/dogami-code/Smart-Contracts-EVM/commit/3b063ec71c18aa313cf614977ca20a18daa0c235]
- StackingLockPeriod.sol [https://github.com/dogami-code/Smart-Contracts-EVM/tree/WIP/Stacking]

**Status:** Fixed

### Classification

**Severity:** Low

**Impact:** 4/5

**Likelihood:** 2/5

### Recommendations

**Recommendation:**

Implement a check to make sure `_timeUnit` is not zero in the reported functions.

**Remediation (revised commit: 2276c97):** A check was implemented to ensure `_timeUnit` is not set to `0`.

## F-2024-0502 - Potential misuse of user funds in reward distribution due to incorrect constructor settings - Low

**Description:** In the current design of the project, a single token is utilized for both staking and distributing rewards. The rewards are allocated to users from a `rewardWallet`, which is specified in the `constructors` of two related contracts. However, there is a potential issue: if the `rewardWallet` is inadvertently set to the contract's own address `address(this)`, it could lead to the misappropriation of funds. This situation could result in using the funds of some users as rewards for others, which is not the intended use of the reward system.

**Assets:**

- StackingFlex.sol [https://github.com/dogami-code/Smart-Contracts-EVM/commit/3b063ec71c18aa313cf614977ca20a18daa0c235]
- StackingLockPeriod.sol [https://github.com/dogami-code/Smart-Contracts-EVM/tree/WIP/Stacking]

**Status:** Fixed

### Classification

**Severity:** Low

**Impact:** 4/5

**Likelihood:** 2/5

### Recommendations

**Recommendation:** Add a check in the constructor to ensure that `rewardWallet` is different than contract address ( `address(this)`).

**Remediation (revised commit: 2276c97):** A check was introduced in the constructor to ensure that `rewardWallet` is different than contract address ( `address(this)`).

## Observation Details

### F-2024-0491 - Floating Pragma - Info

**Description:** The project uses floating pragmas ^0.8.8.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

**Assets:**

- StackingFlex.sol [https://github.com/dogami-code/Smart-Contracts-EVM/commit/3b063ec71c18aa313cf614977ca20a18daa0c235]
- StackingLockPeriod.sol [https://github.com/dogami-code/Smart-Contracts-EVM/tree/WIP/Stacking]

**Status:** Fixed

#### Recommendations

**Recommendation:** Lock the `pragma` version in all contracts as `0.8.8` instead of `^0.8.8`.

**Remediation (revised commit: 2276c97):** The `pragma` version in all contracts was locked to `0.8.19`.

## F-2024-0492 - State variables only set in the constructor should be declared immutable - Info

**Description:**

Compared to regular state variables, the gas costs of constant and immutable variables are much lower. Immutable variables are evaluated once at construction time and their value is copied to all the places in the code where they are accessed.

This will lower the Gas taxes.

The following variables are affected:
`StakingFlex: stakingToken, stakingTokenDecimals, rewardToken, rewardTokenDecimals, rewardWallet.`
`StakingLockPeriod: stakingToken, stakingTokenDecimals, rewardToken, rewardTokenDecimals, rewardWallet, lockPeriodDuration.`

**Assets:**

- StackingFlex.sol [https://github.com/dogami-code/Smart-Contracts-EVM/commit/3b063ec71c18aa313cf614977ca20a18daa0c235]
- StackingLockPeriod.sol [https://github.com/dogami-code/Smart-Contracts-EVM/tree/WIP/Stacking]

**Status:** Fixed

### Recommendations

**Recommendation:**

Consider marking state variables as an `immutable` that never changes on the contract.

**Remediation (revised commit: 2276c97):** The reported state variables were set as `immutable`.

## F-2024-0495 - Missing checks for zero address - Info

**Description:**

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the `zero address`. This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address.

The `Missing zero address control issue` arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.

For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

The zero address should be checked for the following cases:
`StackingLockPeriod: constructor → _stakingToken, _rewardToken, _rewardWallet.`
`StackingFlex: constructor → _stakingToken, _rewardToken, _rewardWallet.`

**Assets:**

- StackingFlex.sol [https://github.com/dogami-code/Smart-Contracts-EVM/commit/3b063ec71c18aa313cf614977ca20a18daa0c235]
- StackingLockPeriod.sol [https://github.com/dogami-code/Smart-Contracts-EVM/tree/WIP/Stacking]

**Status:** Fixed

### Recommendations

**Recommendation:**

It is strongly recommended to implement checks to prevent the zero address from being set during the initialization of contracts. This can be achieved by adding require statements that ensure address parameters are not the zero address.

**Remediation (revised commit: 2276c97):** Zero address checks were implemented for the reported variables.

## F-2024-0503 - Redundant declaration - Info

**Description:** In the `StakingLockPeriod` contract, the state variable `totalStakers` is explicitly initialized to `0` within the `constructor`. This initialization is redundant since variables of type `uint256` in Solidity are automatically initialized to `0` by default. Including this unnecessary declaration not only adds an extra line of code but also incurs a minor, yet avoidable, gas cost during the contract deployment. Removing this redundant initialization can streamline the code for better efficiency and reduce the gas cost associated with the contract's deployment.

**Assets:**

- StackingLockPeriod.sol [https://github.com/dogami-code/Smart-Contracts-EVM/tree/WIP/Stacking]

**Status:** Fixed

### Recommendations

**Recommendation:** It is recommended to remove the aforementioned redundant declaration.

**Remediation (revised commit: 2276c97):** The reported redundant declaration was removed.

## F-2024-0504 - Redundant calculations on claiming rewards - Info

**Description:** The function `_claimRewards` in the contract includes a call to `_calculateRewards`. However, `_claimRewards` is invoked exclusively in scenarios where a user initiates a withdrawal. During such withdrawals, `_calculateRewards` is already called separately to update the user's rewards balance. This results in `_calculateRewards` being executed twice in the same transaction sequence – first independently and then again within `_claimRewards`. This redundancy leads to an unnecessary consumption of Gas, as the same calculations and state updates are performed twice.

**Assets:**

- StackingLockPeriod.sol [https://github.com/dogami-code/Smart-Contracts-EVM/tree/WIP/Stacking]

**Status:** Fixed

### Recommendations

**Recommendation:** It is recommended to remove the redundant call (`_calculateRewards` in `_claimRewards` function),

**Remediation (revised commit: 2276c97):** The redundant call to `_calculateRewards` was removed from `_claimRewards`.

## F-2024-0505 - Checks-Effects-Interactions pattern violation - Info

**Description:**

State variables are updated after the external calls to the token contract.

As explained in Solidity Security Considerations, it is best practice to follow the checks-effects-interactions pattern when interacting with external contracts to avoid reentrancy-related issues.

This best practice is not followed in the following functions:
`StakingFlex: forceWithdraw, _withdraw, _claimRewards.`
`StackingLockPeriod: forceWithdraw, _withdraw, _claimRewards.`

**Assets:**

- StackingFlex.sol [https://github.com/dogami-code/Smart-Contracts-EVM/commit/3b063ec71c18aa313cf614977ca20a18daa0c235]
- StackingLockPeriod.sol [https://github.com/dogami-code/Smart-Contracts-EVM/tree/WIP/Stacking]

**Status:** Fixed

### Recommendations

**Recommendation:**

Follow the checks-effects-interactions pattern when interacting with external contracts, by updating the state variables before making token transfer calls.

**Remediation (revised commit: 2276c97):** The checks-effects-interactions pattern was implemented.

## F-2024-0506 - Redundant calculations in _calculateRewards when time passed is zero - Info

**Description:**

In the `stackingLockPeriod` contract, within the `_calculateRewards` function, there is an inefficient handling of the `startTime` and `endTime` for staking calculations. The `startTime` is set based on a conditional check, which compares `staker.timeOfLastUpdate` with `staker.unlockTime`, choosing the latter if `staker.timeOfLastUpdate` is greater. The `endTime` is invariably set to `staker.unlockTime`. This approach leads to a scenario where both `startTime` and `endTime` can be equal to `staker.unlockTime`, resulting in the calculation of the time passed as zero (`staker.unlockTime - staker.unlockTime = 0`).

Consequently, this results in the function performing calculations using zero as the time elapsed, which is an unnecessary computational step. A more efficient approach would be to directly return zero as the reward in cases where the time passed is calculated to be zero, thus avoiding redundant computations within the `_calculateRewards` function and enhancing the overall efficiency of the contract.

**Assets:**

- StackingLockPeriod.sol [https://github.com/dogami-code/Smart-Contracts-EVM/tree/WIP/Stacking]

**Status:** Fixed

### Recommendations

**Recommendation:**

Consider returning `0` when the condition is met, avoiding unnecessary calculations.

**Remediation (revised commit: 2276c97):** The case where `timeOfLastUpdate` could be greater than `unlockTime` was removed from `_calculateRewards` since it cannot be triggered.

# Disclaimers

## Hacken Disclaimer

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

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

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

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

## Technical Disclaimer

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

# Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

| Severity | Description |
|---|---|
| Critical | Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation. |
| High | High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation. |
| Medium | Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category. |
| Low | Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score. |

# Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

## Scope Details

| | |
|---|---|
| Repository | https://github.com/dogami-code/Smart-Contracts-EVM |
| Commit | 2276c97 |
| Whitepaper | Not provided |
| Requirements | Documentation |
| Technical Requirements | Documentation |

## Contracts in Scope

./src/StakingFlex.sol

./src/StakingLockPeriod.sol