HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Love.io
Date: 18 Oct, 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 Love.io
Auditors	Przemyslaw Swiatowiec | Lead Solidity SC Auditor at Hacken OÜ Kornel Światłowski | SC Auditor at Hacken OÜ Roman Tiutiun | SC Auditor at Hacken OÜ
Tags	Staking
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://love.io
Changelog	18.10.2023 - Initial 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 Love.io (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.

System Overview

LOVE.IO is a flexible staking with a locked period until a planned date. Users can unstake tokens before the planned date but there will be no profit for them in this case. It comes with the following contract:

- *Minting* — a contract that rewards users for depositing their tokens. APY percentages are pre-defined and sorted in ascending order based on time. Users can stake only when the owner supplies liquidity. The contract also implements penalties for early withdrawals.

Privileged roles

- The Owner is responsible to set the *paused* state using *setPaused()* function and has all users` access rights. If the contract is paused, then users can not make stake new tokens.

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

- Documentation Typo in NatSpec. (I03)

Code quality

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

- Style Guide violations found in code.(I01)
- Insufficient Gas modeling.(I02)

Test coverage

Code coverage of the project is **100%** (branch coverage), with a mutation score of 79.31%.

Security score

As a result of the audit, the code contains **2** low 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** The system users should acknowledge all the risks summed up in the risks section of the report.

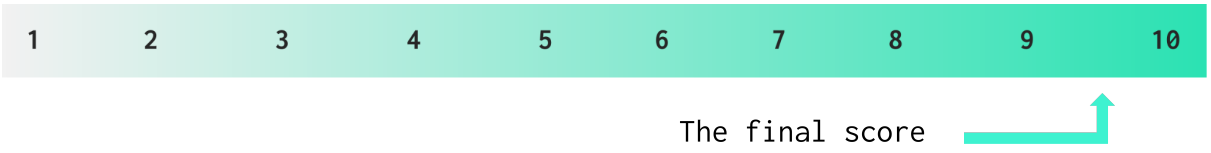

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
18 October 2023	2	0	0	0

Risks

- Stake contract owners have to precisely calculate how many tokens should be allocated for staking rewards. Tokens send to *Minting.sol* contract cannot be withdrawn.

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

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

No medium severity issues were found.

Low

L01. Missing Withdraw Reward Token Mechanism For Contract Owner

Impact	Medium
Likelihood	Low

In the *Minting.sol* contract, owner has to deposit liquidity to enable the contract to operate and pay staking rewards to its users. However, a issue has been identified concerning the contract owner's distribution of reward tokens to users who have staked their tokens.

Example scenarios in which this issue arises:

- The owner transfers more tokens than required.
- Users choose not to utilize the contract.
- Owners seek to recover penalties.

In situations like these, when a withdrawal mechanism is not present, tokens can potentially become trapped within the contract. While the Love protocol documentation does touch on this mechanism, it is considered a best practice for security to enable contract owners to withdraw their tokens in the event of unforeseen circumstances, specifically the portion that is not already committed to active liquidity earmarked for rewards.

Path: ./contracts/Minting.sol

Recommendation: It is recommended to add a withdrawal mechanism for reward tokens that are not part of an active liquidity (*maxPotentialDebt*).

Found in: 9f9b2b4

Status: New

L02. Missing validation If Stake Token Supports Fee-On-Transfer

Impact	Medium

Likelihood	Low

According to the technical requirements, the contract is not designed to accommodate fee-on-transfer tokens. Regrettably, this limitation lacks validation in the code.

The reward token is included in the *constructor()* without undergoing any validation process. Consequently, there exists a risk that the stake contract owner could inadvertently introduce an unsupported fee-on-transfer token, potentially resulting in an erroneous state of the contract.

Path: ./contracts/Minting.sol: constructor();

Recommendation: Implementing validation for the reward token is strongly recommended to ensure that only tokens not supporting fee-on-transfer are added. This can be achieved by introducing a new function that accepts liquidity from the owner and verifies that the transferred amount matches the deposited amount. Alternatively, this validation can be integrated into every stake transaction to maintain the integrity of the contract.

Found in: 9f9b2b4

Status: New

Informational

I01. Style Guide Violation - Order of Layout

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each contract, library, or interface is as follows:

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

Functions should be ordered and grouped by their visibility as follows:

- Constructor
- Receive function (if exists)
- Fallback function (if exists)

- External functions
- Public functions
- Internal functions
- Private functions

Within each grouping, *view* and *pure* functions should be placed at the end.

In the *Minting.sol* contract, the order of external, public, internal, and private functions is not followed, and the functions are not grouped according to their visibility.

Path: ./contracts/Minting.sol

Recommendation: Consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts.

Found in: 9f9b2b4

Status: New

I02. Use Custom Errors Instead Of Error Strings To Save Gas

Custom errors were introduced in Solidity version 0.8.4, and they offer several advantages over traditional error handling mechanisms:

1. Gas Efficiency: Custom errors can save approximately 50 gas each time they are hit because they avoid the need to allocate and store revert strings. This efficiency can result in cost savings, especially when working with complex contracts and transactions.
2. Deployment Gas Savings: By not defining revert strings, deploying contracts becomes more gas-efficient. This can be particularly beneficial when deploying contracts to reduce deployment costs.
3. Versatility: Custom errors can be used both inside and outside of contracts, including interfaces and libraries. This flexibility allows for consistent error handling across different parts of the codebase, promoting code clarity and maintainability.

Path: ./contracts/Minting.sol

Recommendation: To save gas, it's recommended to use custom errors instead of strings.

Found in: 9f9b2b4

Status: New

I03. Typos In The Code

The comments in the *Minting* contract require the following corrections:

- In the comment for the *numOfActiveStakes* variable, it should be 'currently' not 'currenlty'.
- In the comment within the *withdraw()* function, it should be 'contract' not 'comtract'.
- In the same comment within the *withdraw()* function, it should be 'subtract' not 'substract'.
- In the NatSpec for the *_calculateRewardForDurationAndStakingPeriod()* function, it should be 'subtracting' not 'substracting.'

Path: ./contracts/Minting.sol

Recommendation: Fix aforementioned typos.

Found in: 9f9b2b4

Status: New

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

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.

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

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

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 won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

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

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

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

Appendix 2. Scope

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

Initial review scope

Repository	https://github.com/gotbitlabs/love-staking
Commit	9f9b2b44598bf0d38c9b97e2d24ff3e29d0fb1a7
Whitepaper	Not provided
Requirements	File: Love Minting PSRS.docx SHA3: fd61bd186b706a83455c4a8ad296383a2309860c22b496167b9fdc96b71eb61a
Technical Requirements	File: https://docs.google.com/document/d/1JMwjPNY1B18LO5sz0cHBS_VHyWQIrCZIXCRlV1UnJCc/edit?usp=sharing SHA3: 073c88fca5f10bd05193ad3a95e845561585f76bb4b310b84724fec20b56341d
Contracts	File: contracts/Minting.sol SHA3: dacc803b98d939138e3489e1088d8a5f317e924249ad9c982241fe09748d367f