Seascape

Seascape Lighthouse Security Analysis

by Pessimistic

This report is public

October 20, 2021

Abstract

In this report, we consider the security of smarts contracts of Seascape Lighthouse project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

Summary

In this report, we considered the security of Seascape Lighthouse smart contracts. We performed our audit according to the procedure described below.

The initial audit showed two critical issues: Non-mintable tokens and Reentrancy; several issues of medium severity including Bug, Access control, Discrepancy with the documentation, Insufficient replay protection, and others. Also, several issues of low severity were found.

After the initial audit, the code base was updated. In this update, all the critical issues and most of the medium severity issues were fixed.

General recommendations

We recommend limiting the powers of staker role, fixing discrepancy with the documentation and tests issues, and following CEI-pattern. We also recommend adding CI to run tests, calculate code coverage, and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Seascape Lighthouse project on a public GitHub repository, commit 4523d29a7bd8978075b3b80d046cd268f5ea7d7a.

The project has public documentation, the code is partially covered with NatSpec comments.

Optimization is not turned on in the config.

Tests do not run.

The total LOC of audited sources is 771.

Code base update

After the initial auidt, the code base was updated. For the recheck, we were provided with commit 2b8f9463ab619fb7fbd29d4f9d5cab96c281a31c.

In this update, most of the issues were fixed.

Also, the documentation was improved with this update.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan project's code base with automated tools: Slither and SmartCheck.
  - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
  - We manually analyze code base for security vulnerabilities.
  - We assess overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They often lead to the loss of funds or other catastrophic failures. The contracts should not be deployed before these issues are fixed. We highly recommend fixing them.

Non-mintable tokens (fixed)

In **LighthouseNft** contract, `mint()` function will always revert due to the check at line 52 since `projectId` storage variable is never initialized.

As a result, users do not receive NFT for their investments.

The issue has been fixed and is not present in the latest version of the code.

Reentrancy (fixed)

An attacker can perform reentrancy attack to mint multiple NFTs for a signle deposit. They can later burn those NFTs to receive more tokens than allowed, i.e. steal them from other users or the project.

Reentrancy attack is possible, sinse the `mint()` function of **LighthouseMint** contract violates CEI (checks-effects-interactions) pattern:

- `mint()` relies on `ERC721._safeMint` function from OpenZeppelin, which perfroms an external call to the token recepient.
- `mintedNfts[projectId][msg.sender]` storage variable is updated at line 99 after the external call.

An example of external call chain: `LighthouseMint.mint` → `LighthouseNft.mint` → `IERC721Receiver(attacker).onERC721Received` → `LighthouseMint.mint`

We recommend always following CEI pattern. We also advise to use `safe` functions of ERC721 implementation from OpenZeppelin with great caution, since they perfrom external call.

The issue has been fixed and is not present in the latest version of the code.

Medium severity issues

Medium issues can influence project operation in current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

Bug (fixed)

`getTierLevel()` function of **LighthouseTier** contract has an inverted condition inside `if` block at line 179: should be `tiers[investor].usable` instead of `!tiers[investor].usable`.

The issue has been fixed and is not present in the latest version of the code.

Access control (fixed)

In **LighthouseProject** contract, `transferPrefund()` function should only be accessed by the owner of the contract. However, anyone can call it.

The issue has been fixed and is not present in the latest version of the code.

Discrepancy with the documentation

The documentation states:

```
The usage of Tier resets its level. According to plan, user can use
his Tier only once per project. After that an investor has to
reclaim it.
```

However, in the code, there are deviations from this behavior:

- In **LighthousePrefund** contract, `prefund()` function does not reset user's tier after the investment is made.

  The issue has been fixed and is not present in the latest version of the code.

- In **LighthouseAuction** contract, `participate()` function does not reset user's tier after the investment, and does not check if user's tier is valid.

Additionally, `claim()` and `use()` functions of **LighthouseTier** contract behave inconsistently:

- The `claim()` function allows user to increase their tier by one level or reclaim their used tier.
- The `use()` function effectively disables reclaim feature, since the function always resets user's tier to zero level.

The issues have been fixed and are not present in the latest version of the code.

Insufficient replay protection (fixed)

Consider adding `chanId` and address of the called contract to every signed message, so the signature can not be in reused for other contracts of the project on the same or different chain.

The issue has been fixed and is not present in the latest version of the code.

Overpowered roles

The owner and staker roles have excessive powers:

- The owner can change the address of the project's token at any time by calling `setPcc()` function of **LighthouseProject** contract, since values of `usedPccs` mapping are never set.

  The issue has been fixed and is not present in the latest version of the code.

- The owner and the staker can transfer PCC tokens from the contract address to any recipient in **LighthouseBurn** contract at lines 62 and 75.

- The owner can modify the parameters of the system that affect the operation of the project.

In the current implementation, the system depends heavily on the owner of the contract. Therefore, there are scenarios that may lead to undesirable consequences for the project and its users, e.g., if the owner's or stakers' private keys become compromised. We recommend designing contracts in a trustless manner or at least using proper key management system, e.g., multisig.

> *Comment from developers: Multisig is used as the owner.*

ERC-20 standard violation (fixed)

ERC-20 standard states:

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned!
```

However, returned values from `transferFrom()` calls are not checked.

The issue has been fixed and is not present in the latest version of the code.

Tests issues

The project has tests. However, the tests do not run. Testing is crucial for code security and audit does not replace tests in any way.

We highly recommend covering the code with tests and making sure that the code coverage is sufficient. We also recommend adding functional tests that fully cover project's lifecycle in the system.

Low severity issues

Low severity issues do not directly affect project's operations. However, they might lead to various problems in the future versions of the code. We recommend taking them into account.

Code quality

- In **LighthouseProject** contract, checks `startTime > 0` and `endTime > 0` are redundant.

  The issues have been fixed and are not present in the latest version of the code.

- In **LighthouseTier** contract, `investAmount` mapping is not used. Also, it is declared as `uint8[3]`, though its type should be `uint256[3]`.

  The issue has been fixed and is not present in the latest version of the code.

- Consider performing multiplication operations before division to improve calculation precision at lines 407 and 412 of **LighthouseProject** contract.

  The issues have been fixed and are not present in the latest version of the code.

- We highly recommend following CEI (checks-effects-interactions) pattern since it prevents some types of re-entrancy attacks. Consider updating storage variables before external calls in:
  - **LighthouseTier** contract at line 142.

    The issue has been fixed and is not present in the latest version of the code.
  - **LighthousePrefund** contract at line 95.
  - **LighthouseAuction** contract at line 76.
  - **LighthouseBurn** contract at lines 72, 83, 140 and 142.
  - **LighthouseNft** contract at line 57.

- Consider using ECDSA for stronger replay protection in:
  - **LighthousePrefund** contract at line 77.
  - **LighthouseTier** at line 133.
  - **LighthouseAuction** at line 69.

- Storage variables are `internal` by default. Consider declaring visibility of `fundCollector` variable in **LighthousePrefund** contract explicitly to improve code readability.
- Non-negative check of `level` parameter at lines 117 and 155 in **LighthouseTier** contract is redundant since `uint8` type can not hold negative values.
- Consider declaring functions as `external` where possible. It improves code readability and optimizes gas consumption.

Insufficient documentation (fixed)

The project has a documentation. However, it does not explain the details of the code operation, main design choices, and overall project structure.

Considering the importance of the contracts to the project, the detailed documentation is required to streamline both development and audit processes. It should explicitly explain the purpose and behavior of the contracts, their interactions, and main design choices.

The documentation was improved.

Dependency management

Some tools are in `dependencies` section of **package.json** file rather than in `devDependencies`.

Notes

Limited Crowns supply

Tokens are first burned during the auction and then can be burned if the user chooses to burn pcc tokens in `burnForPcc` function of **LighthouseBurn** contract at line 114. Since the number of crowns tokens is limited, they can end quickly if the system is used extensively.

Logic

It is not clear from the code at line 401 of **LighthouseProject** contract whether the case is really impossible when the number of invested tokens is greater than or equal to the maximum number of tokens for a given level.

ERC20 standard violation

CrownsToken contract does not follow ERC-20 standard. However, it is out of scope of this audit.

This analysis was performed by Pessimistic:

Daria Korepanova, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

October 20, 2021