Audit Report

Hadron Labs Lido Satellite

v1.0

July 29, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Hadron Labs to perform a security audit of the Lido Satellite CosmWasm smart contract.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/hadronlabs-org/lido-satellite
Commit	`358fd115fc60b8fbd5a2c13ba9c0b85c88f7d65c`
Scope	All code was in scope.
Fixes verified at commit	`57c27452a1ea592a5fb469b5ce7286fedf33bc13` Note that changes to the codebase beyond fixes after the initial audit have not been in scope of our fixes review.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The contract in scope of this review allows users to mint canonical `tokenfactory` tokens via sending bridged funds to the contract with `ExecuteMsg::Mint`. To redeem funds, users can simply send canonical funds to the contract with `ExecuteMsg::Burn`.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Low**	-
Code readability and clarity	**High**	-
Level of documentation	**High**	-
Test coverage	**High**	`cargo tarpaulin` reports a test coverage for the contracts in scope of 97.01% (65/67 lines covered).

Summary of Findings

No	Description	Severity	Status
1	Missing validation of the `bridged_denom` format	Minor	Resolved
2	Additional funds sent by the user besides `config.bridged_denom` are locked	Minor	Resolved
3	Missing check that the contract has enough funds to cover the `creator_fee`	Informational	Acknowledged
4	Overflow checks not enabled for release profile	Informational	Acknowledged
5	“Migrate only if newer” pattern is not followed	Informational	Acknowledged
6	Contract name is hardcoded	Informational	Resolved
7	The full `tokenfactory` denom can be derived programmatically instead of supplied manually	Informational	Resolved

Detailed Findings

1. Missing validation of the `bridged_denom` format

Severity: Minor

In `contracts/lido-satellite/src/msg.rs:16-21`, the `validate` function does not validate the format of the `bridged_denom` variable.

This is problematic since the contract is designed to hold only funds denominated in bridged denoms.

Specifically, the provided denom should adhere to the required format of `ibc/{denom}` before proceeding with any subsequent processing steps.

Recommendation

We recommend validating the `bridged_denom` structure to ensure its correctness. If it is incorrect, an appropriate error message should be returned.

Status: Resolved

2. Additional funds sent by the user besides `config.bridged_denom` are locked

Severity: Minor

In `contracts/lido-satellite/src/execute.rs:15` and `39`, the `find_denom` function checks for the existence of a `Coin` with the `config.bridged_denom` among the funds sent with the message.

This validation does not ensure however that no other native tokens have been sent, and any such additional native tokens are not returned to the user, so they will be stuck in the contract forever.

While blockchains generally do not protect users from sending funds to wrong accounts, reverting extra funds increases the user experience.

Recommendation

We recommend checking that the transaction contains only the expected `Coin` and no additional native tokens using https://docs.rs/cw-utils/latest/cw_utils/fn.must_pay.html.

Status: Resolved

3. Missing check that the contract has enough funds to cover the `creator_fee`

Severity: Informational

In `contracts/lido-satellite/src/contract.rs:31`, before creating a new denom, no check is performed to ensure that the contract has sufficient funds to cover the `creator_fee`.

Since the `tokenfactory` module imposes a fee for denom creation in https://github.com/neutron-org/neutron/blob/v1.0.2/x/tokenfactory/keeper/createdenom.go#L15, if the contract does not hold enough funds, an error will be raised.

This behavior could complicate the contract deployment process and lead to inefficiencies.

Recommendation

We recommend checking whether the contract possesses or the instantiator provided sufficient funds to cover the `creator_fee`.

Status: Acknowledged

4. Overflow checks not enabled for release profile

Severity: Informational

The `lido-satellite` contract does not enable `overflow-checks` for the release profile.

While enabled implicitly through the workspace manifest, future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Acknowledged

5. “Migrate only if newer” pattern is not followed

Severity: Informational

The contract within the scope of this audit is currently migrated without regard to its version. This can be improved by adding validation to ensure the migration is only performed if the supplied version is newer.

Recommendation

We recommend following the migrate “only if newer” pattern defined in the CosmWasm documentation.

Status: Acknowledged

6. Contract name is hardcoded

Severity: Informational

The contract name `CONTRACT_NAME` supplied to the `set_contract_version` function when initializing the contract is hardcoded to `”crates.io:lido-satellite”` in `contracts/lido-satellite/src/contract.rs:12`.

To follow best practices, the contract name should be used from the `Cargo.toml`.

Recommendation

We recommend using `env!("CARGO_PKG_NAME")` to provide the name of the contract.

Status: Resolved

7. The full `tokenfactory` denom can be derived programmatically instead of supplied manually

Severity: Informational

Minting and burning funds necessitates using the full `tokenfactory` denom, which is retrieved through the `get_full_tokenfactory_denom` function defined in `contracts/lido-satellite/src/state.rs:11-17`.

This function always assembles the full denom in the format `factory/{contract_address}/{canonical_subdenom}`, where `canonical_subdenom` is a configuration variable set during contract initialization, and `contract_address` represents the contract's address.

This approach can be improved on by querying the denom after its creation instead of using the hardcoded denom format, thereby relying less on the internal implementation of Neutron's `tokenfactory` module.

Recommendation

We recommend using a submessage for the `NeutronMsg::CreateDenom` message and implementing a reply handler to retrieve the full `tokenfactory` denom via the `query_full_denom` function provided by the Neutron SDK.

Status: Resolved