Security Audit Report

Drop Initia Liquidity Provider

v1.0

April 17, 2025

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Droplet Labs Ltd to perform a security audit of Drop Initia Liquidity Provider.

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/drop-contracts
Commit	`0ba7ebb9341befa9aeb903af6b8ea91e76f45ac1`
Scope	The scope is limited to the Move contract located in the `movevm/liquidity-provider` directory and its integration with the Drop codebase.
Fixes verified at commit	`fd417384aa80bf9c9249200914808a642fe2fa22` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

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

Drop is a liquid staking protocol designed for deployment on the Neutron chain within the Cosmos ecosystem.

The protocol leverages Inter-Blockchain Communication (IBC), Interchain Accounts (ICA) and Interchain Queries (ICQ) to facilitate seamless staking and unstaking across the Cosmos ecosystem.

Drop integrates an auto-compounding feature, automatically restaking rewards to optimize yield.

The audit scope is restricted to the Liquidity Provider contract for the Initia chain and its integration with the Drop codebase.

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**, **Partially Resolved**, 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-Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium**	The client shared a high-level overview of the contract.
Test coverage	**Low**	No tests have been implemented.

Summary of Findings

No	Description	Severity	Status
1	Unrestricted callback execution allows for price check bypass	Critical	Resolved
2	Insecure Oracle failure handling logic	Critical	Partially Resolved
3	Incorrect computation disables safety guards against MEV	Critical	Partially Resolved
4	The `backup` function allows centralized fund withdrawal	Minor	Partially Resolved
5	Hardcoded `min_liquidity_amount` could lead to unsafe liquidity provision	Minor	Acknowledged
6	Inefficiency in Oracle price updates	Informational	Resolved
7	Missing event emissions	Informational	Resolved
8	Use of magic numbers decreases maintainability	Informational	Resolved
9	Unfinished development	Informational	Resolved

Detailed Findings

1. Unrestricted callback execution allows for price check bypass

Severity: Critical

The intended flow for providing liquidity begins with the `provide` function, which sends a Stargate transaction to trigger a price update.

Once the transaction completes, a `callback` function is executed to verify its success, ensure the price is up-to-date, and confirm that it aligns with the defined limits.

However, the `callback` function lacks any access control mechanisms, allowing unauthorized users to invoke it with arbitrary input arguments.

This vulnerability permits malicious actors to bypass the intended flow entirely and directly provide liquidity, circumventing the intended protocol safeguards.

Recommendation

We recommend implementing an access control mechanism that would ensure the `callback` function can be executed only by the contract itself.

Status: Resolved

2. Insecure Oracle failure handling logic

Severity: Critical

The `callback` function is designed to execute upon completion of a Stargate transaction, which involves retrieving a price update from the Oracle.

When the Oracle responds successfully, the `liquidity_provider` module verifies that the price is current and ensures the slippage is within acceptable limits before allowing liquidity provision.

However, if the Oracle fails to respond or lacks data on the desired price, the callback function is still executed with the `success` parameter set to `false`. Despite this failure, the module proceeds to execute the `provide_liquidity` function without any price verification.

Additionally, due to the issue described in "Unrestricted callback execution allows for price check bypass", attackers can arbitrarily provide `false` for the `success` parameter to disable price verification.

This behavior exposes users of the module to significant monetary risks, as liquidity can be provided under unverified or outdated pricing conditions whenever the oracle fails.

Recommendation

We recommend changing the implementation so that the liquidity is not provided when `callback` is executed with `success` parameter set to `false`. Returning an error or otherwise informing the user about why the liquidity was not provided is also highly recommended.

Status: Partially Resolved

3. Incorrect computation disables safety guards against MEV

Severity: Critical

In `movevm/liquidity-provider/sources/poc.move:140`, the `callback` function contains a logic error in its pool price validation mechanism.

The intended functionality is to ensure that the pool price does not deviate by more than 1% from the oracle's price.

However, due to a calculation error, the ratio is incorrectly set to `101/1`, equating to a 10,100% deviation threshold instead of the expected 1%.

This oversight effectively disables the intended security measures designed to protect against Maximum Extractable Value (MEV) and sandwich attacks.

Furthermore, the static 1% maximum slippage threshold may become overly restrictive during periods of high market volatility, leading to frequent transaction failures.

Status: Partially Resolved

We recommend correcting the ratio calculation to properly reflect the intended 1% threshold.

Moreover, we recommend implementing configurable slippage parameters to specify the acceptable slippage tolerance when interacting with the DEX module.

4. The `backup` function allows centralized fund withdrawal

Severity: Minor

In `movevm/liquidity-provider/sources/poc.move:72-84`, the `backup` function is implemented as a last-resort mechanism, allowing the module administrator to withdraw all funds of a specific coin denomination from the module's object address. This design is intended to safeguard funds in the event of an unrecoverable bug.

However, it introduces a centralization risk, as in the case of compromised private keys of the module admin, the attacker has unilateral authority to withdraw all held funds.

Recommendation

We recommend implementing additional safeguards to mitigate this centralization risk, such as introducing a multi-signature scheme to authorize the execution of the backup function.

Status: Partially Resolved

The client implemented a configurable `backup` address that could be set as a multi-signature account.

5. Hardcoded `min_liquidity_amount` could lead to unsafe liquidity provision

Severity: Minor

In `movevm/liquidity-provider/sources/poc.move:93`, the `dex::single_asset_provide_liquidity_script` function is called with `min_liquidity_amount` parameter set to `None` during liquidity provision operations.

This is problematic because even with accurate Oracle price data, users could receive substantially fewer tokens than expected due to potential pool liquidity draining.

Recommendation

We recommend implementing optional minimum liquidity amount parameters for liquidity provision operation as a user-controllable variable.

Status: Acknowledged

6. Inefficiency in Oracle price updates

Severity: Informational

In `movevm/liquidity-provider/sources/poc.move:107`, the `store` function retrieves the price from the Oracle and saves it within the module.

However, updating the price multiple times within the same block is redundant and inefficient.

If the stored timestamp is already equal to the current block's timestamp, the `get_price` operation should be skipped to optimize performance and reduce unnecessary Oracle interactions.

Recommendation

We recommend adding a conditional check within the `store` function to compare the stored timestamp with the current block timestamp. If they are identical, the `get_price` operation should be bypassed.

Status: Resolved

7. Missing event emissions

Severity: Informational

In the whole contract, particularly in administrative functions like `backup` and core functionality like liquidity provision, an event emissions mechanism is absent. This omission significantly impacts the contract's transparency and makes it difficult to track important state changes and actions.

The lack of events, especially in administrative functions, reduces the ability to monitor privileged actions and perform off-chain tracking of critical operations like emergency fund withdrawals.

Recommendation

We recommend implementing detailed event emissions across all significant state-changing functions, especially for administrative operations, like the `backup` function, and core functionalities, like liquidity provision.

Status: Resolved

8. Use of magic numbers decreases maintainability

Severity: Informational

In `movevm/liquidity-provider/sources/poc.move:68`, a hard-coded number literal without context or a description is used. Using such “magic numbers” goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase.

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Resolved

9. Unfinished development

Severity: Informational

In `movevm/liquidity-provider/Move.toml`, configuration addresses for the contract owner and LP token recipient are currently set as placeholder values.

These placeholder values indicate an incomplete development configuration that requires finalization before deployment.

Recommendation

We recommend completing the configuration by setting appropriate addresses for the contract owner and LP token recipient roles.

Status: Resolved