AUDIT REPORT

Lion Dao

Savanna Contracts

Prepared by SCV-Security

On 13th March 2024

Table of Contents

Table of Contents **2**
Introduction **3**
Scope Functionality 3
Submitted Codebase 6
Revisions Codebase 6
Methodologies 6
Code Criteria 7
Findings Summary **8**
Findings Technical Details **9**
1. Lockdrop account can permissionless execute arbitrary messages 9
2. Users can bypass lock functionality 10
3. Migrate only if newer pattern should be implemented 11
4. Remove empty execute file 12
5. Remove unused fields and enums 13
6. Remove redundant error 14
7. ura dependency pointing to latest main git branch 15
8. Lockdrop account unimplemented query 16
9. Commented code 17
10. Multiple todo! macro 18
Document Control **19**
Appendices **20**
A. Appendix - Risk assessment methodology 20
B. Appendix - Report Disclaimer 21

Introduction

SCV has been engaged by Lion Dao to conduct a comprehensive security review with the goal of identifying potential security threats and vulnerabilities within the codebase. The purpose of this audit is to evaluate the security posture of the codebase and provide actionable recommendations to mitigate any identified risks. This report presents an overview of the findings from our security audit, outlining areas of concern and proposing effective measures to enhance the codebase's security.

Scope Functionality

The Savanna protocol provides a framework for creating and interacting with lockdrop contracts. These contracts allow participants to temporarily deposit and lock certain tokens and receive reward tokens in return. The rewards are variable depending on the amount and duration of the user's token lock, incentivizing long-term engagement and investment in the project.

The lifecycle of a lockdrop contract after creation follows the phases below.

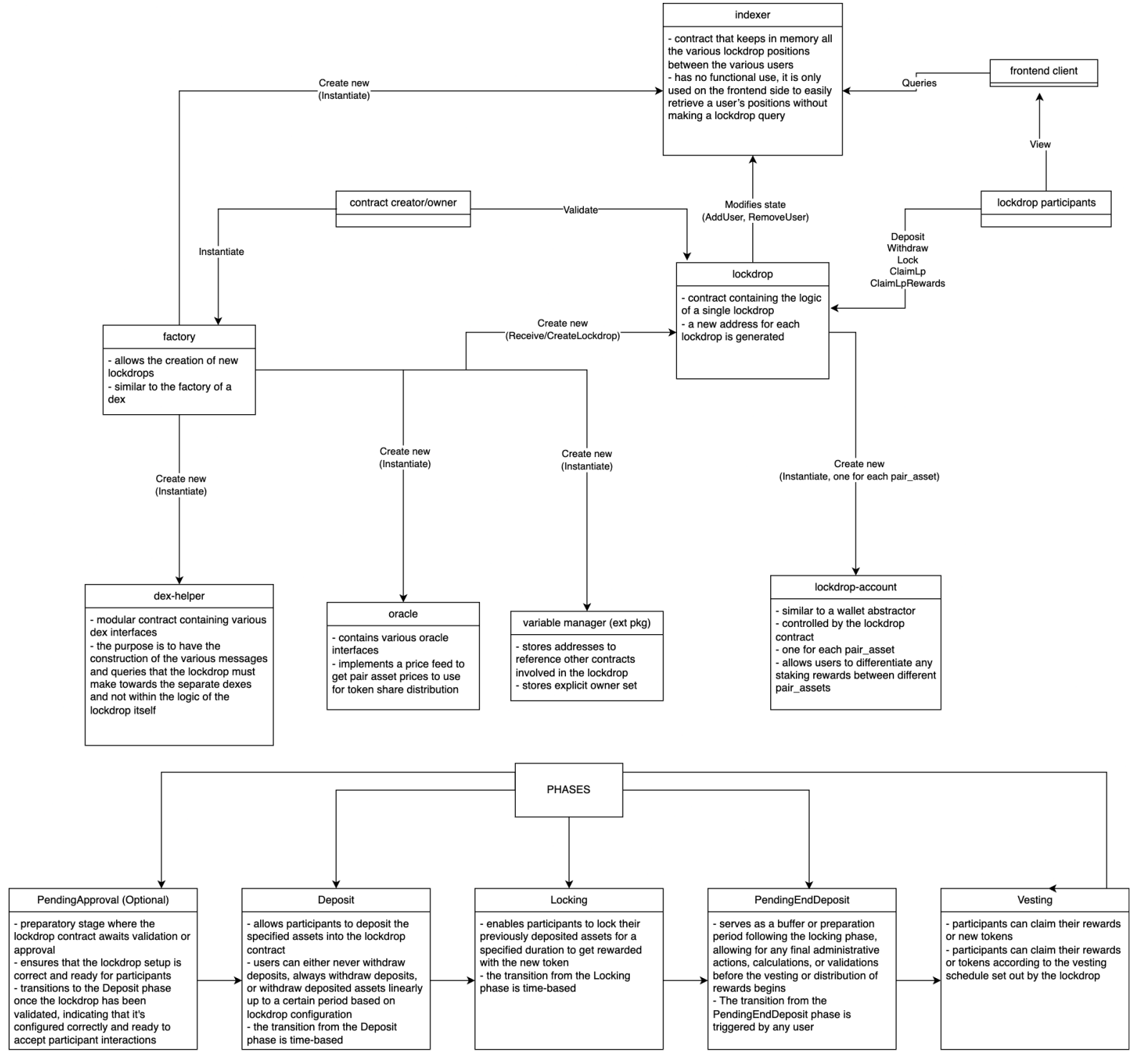
1. **PendingApproval (optional)**
    a. Awaits validation from an admin to ensure that the lockdrop is properly configured
2. **Deposit**
    a. Allows users to deposit their pair assets into the lockdrop. Once deposited, the users can conditionally withdraw their pair assets based on the lockdrop configuration
3. **Locking**
    a. Allows users to commit their pair assets for a specified duration to qualify for rewards
4. **PendingEndDeposit**
    a. Transition phase to calculate asset prices and allocate distribution shares for the new token
5. **Vesting**

a. Distributes new tokens to users based on their locked stakes. The vesting schedule is defined in the lockdrop configuration

Additionally, the protocol is defined through the following contracts:

- `factory`: acts as a contract generator, creating new instances of the following contracts with customizable parameters to suit different distribution goals and token types
- `lockdrop`: handles the locking of tokens, calculation of rewards, and distribution of new tokens or rewards to participants
- `lockdrop-account`: manages individual participant accounts within each lockdrop, holding locked tokens and accrued rewards, and segregating participant interactions for enhanced security and modularity
- `dex-helper`: provides an interface for interacting with various dexes to facilitate liquidity pool creation and liquidity provision
- `oracle`: provides an interface for interacting with oracles to provide current token prices to accurately calculate token values and reward distribution
- `indexer`: used directly by the frontend client to get the current lockdrop positions for a given user

For more context, SCV-Security is providing the following diagram:


indexer
- contract that keeps in memory all the various lockdrop positions between the various users
- has no functional use, it is only used on the frontend side to easily retrieve a user's positions without making a lockdrop query
frontend client
Queries
Create new (Instantiate)
View
contract creator/owner
Validate
Modifies state (AddUser, RemoveUser)
lockdrop participants
Instantiate
Deposit
Withdraw
Lock
ClaimLp
ClaimLpRewards
lockdrop
- contract containing the logic of a single lockdrop
- a new address for each lockdrop is generated
factory
- allows the creation of new lockdrops
- similar to the factory of a dex
Create new (Receive/CreateLockdrop)
Create new (Instantiate)
Create new (Instantiate)
Create new (Instantiate)
Create new (Instantiate, one for each pair_asset)
dex-helper
- modular contract containing various dex interfaces
- the purpose is to have the construction of the various messages and queries that the lockdrop must make towards the separate dexes and not within the logic of the lockdrop itself
oracle
- contains various oracle interfaces
- implements a price feed to get pair asset prices to use for token share distribution
variable manager (ext pkg)
- stores addresses to reference other contracts involved in the lockdrop
- stores explicit owner set
lockdrop-account
- similar to a wallet abstractor
- controlled by the lockdrop contract
- one for each pair_asset
- allows users to differentiate any staking rewards between different pair_assets
PHASES
PendingApproval (Optional)
- preparatory stage where the lockdrop contract awaits validation or approval
- ensures that the lockdrop setup is correct and ready for participants
- transitions to the Deposit phase once the lockdrop has been validated, indicating that it's configured correctly and ready to accept participant interactions
Deposit
- allows participants to deposit the specified assets into the lockdrop contract
- users can either never withdraw deposits, always withdraw deposits, or withdraw deposited assets linearly up to a certain period based on lockdrop configuration
- the transition from the Deposit phase is time-based
Locking
- enables participants to lock their previously deposited assets for a specified duration to get rewarded with the new token
- the transition from the Locking phase is time-based
PendingEndDeposit
- serves as a buffer or preparation period following the locking phase, allowing for any final administrative actions, calculations, or validations before the vesting or distribution of rewards begins
- The transition from the PendingEndDeposit phase is triggered by any user
Vesting
- participants can claim their rewards or new tokens
- participants can claim their rewards or tokens according to the vesting schedule set out by the lockdrop

Submitted Codebase

Savanna Contracts	
Repository	https://github.com/BIG-Labs/savanna-contracts
Commit	`b18092fbb6cda22cc5932b4c8c55badc853c355f`
Branch	`main`

Revisions Codebase

Savanna Contracts	
Repository	https://github.com/BIG-Labs/savanna-contracts
Commit	`8c37a09b71b745493d4ba60612699c65fa3935dc`
Branch	`main`

Methodologies

SCV performs a combination of automated and manual security testing based on the scope of testing. The testing performed is based on the extensive experience and knowledge of the auditor to provide the greatest coverage and value to Lion Dao. Testing includes, but is not limited to, the following:

- Understanding the application and its functionality purpose.
- Deploying SCV in-house tooling to automate dependency analysis and static code review.
- Analyse each line of the code base and inspect application security perimeter.
- Review underlying infrastructure technologies and supply chain security posture.

Code Criteria

This section provides an evaluation of specific criteria aspects as described below:

- **Documentation:** Evaluating the presence and comprehensiveness of publicly available or provided explanatory information, diagram flowcharts, comments, and supporting documents to enhance code understanding.
- **Coverage:** Evaluating whether the code adequately addresses all necessary cases and scenarios, ensuring that the intended functionality or requirements are sufficiently covered.
- **Readability:** Assessing how easily the code can be understood and maintained, considering factors such as code structure, naming conventions, and overall organisation.
- **Complexity:** Evaluating the complexity of the code, including factors such as, number of lines, conditional statements, and nested structures.

The status of each criteria is categorised as either **SUFFICIENT** or **NOT-SUFFICIENT** based on the audit assessment. This categorisation provides insights to identify areas that may require further attention and improvement.

Criteria	Status	Notes
Documentation	**SUFFICIENT**	The savanna contracts did not provide detailed documentation. But the team provided architecture details and answers to all of the audit team's questions.
Coverage	**SUFFICIENT**	Testing coverage is considered sufficient with 86.53% of the code currently covered.
Readability	**SUFFICIENT**	The codebase had good readability overall and utilised many Rust and CosmWasm best practices.
Complexity	**SUFFICIENT**	The contracts within the scope of this audit were relatively complex due to the multi-contract interaction and detailed state.

Findings Summary

Summary Title	Risk Impact	Status
Lockdrop account can permissionless execute arbitrary messages	SEVERE	RESOLVED
Users can bypass lock functionality	MODERATE	RESOLVED
Migrate only if newer pattern should be implemented	INFO	ACKNOWLEDGED
Remove empty execute file	INFO	RESOLVED
Remove unused fields and enums	INFO	RESOLVED
Remove redundant error	INFO	RESOLVED
ura dependency pointing to latest main git branch	INFO	RESOLVED
Lockdrop account unimplemented query	INFO	RESOLVED
Commented Code	INFO	RESOLVED
Multiple “todo!” macro	INFO	RESOLVED

Findings Technical Details

1. Lockdrop account can permissionless execute arbitrary messages

RISK IMPACT: SEVERE	STATUS: RESOLVED

Description

The `Execute` message of the `lockdrop-account` contract does not ensure that the sender is the owner of the contract. This will allow any address to execute any arbitrary message from the `lockdrop-account` contract.

Recommendation

We recommend adding a validation to the execute message of the contract. Additionally, we recommend creating a whitelist of message types that the `lockdrop-account` contract can only execute.

2. Users can bypass lock functionality

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

The `run_lock` function in `contracts/lockdrop/src/execute.rs:194` allows a depositor to lock a specified amount of their deposit by specifying a `LockMsg`. The function currently handles the case of an existing user lock by calling `undo_lock` and starting the lock creation with a fresh state. It does this by reducing the `config.locked_assets_multiplier` by the existing locked amount and deleting `user.locked_assets` and `user.locked_assets_multiplier`. Then the function proceeds to calculate the new locked value based on the amount and duration and finally updates `USERS_DEPOSIT` and `CONFIG`.

This is problematic because a user can abuse this mechanism by specifying an amount in `LockMsg` that is less than their current locked amount, and it will effectively unlock the difference between the two. Large holders can take advantage of this mechanic to manipulate locked amount and potentially dissuade smaller users from locking, thus receiving more share for themselves.

Recommendation

We recommend ensuring that users cannot specify an amount and time less than their current lock when calling `run_lock`.

3. Migrate only if newer pattern should be implemented

RISK IMPACT: INFORMATIONAL	STATUS: ACKNOWLEDGED

Description

The contracts within the scope of this audit implement default migration messages. It is best practice to implement a migrate only if a newer pattern as well as implementing detailed attributes.

Recommendation

We recommend implementing a migrate only if a newer pattern for the savanna contracts.

4. Remove empty execute file

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The `execute.rs` file in the `dex-helper` directory is empty. All of the execute logic is included in the corresponding `contract.rs` file.

Recommendation

We recommend removing the empty `execute.rs` file.

5. Remove unused fields and enums

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The `DexNotHanlded` contract error enum created in `contracts/lockdrop/src/response.rs:L15` is never used. Additionally, there are multiple fields named `_msg` that are not used in their corresponding functions in the following files:

- `contracts/lockdrop/src/execute.rs:67`
- `contracts/lockdrop/src/execute.rs:458`
- `contracts/lockdrop/src/execute.rs:703`
- `contracts/lockdrop-account/src/contract.rs:15`

Recommendation

We recommend removing the unused `_msg` fields and either removing the `DexNotHanlded` error or renaming the error to `DexNotHandled` and throwing it when validating each unique dex in `/contracts/lockdrop/src/contract.rs:99`.

6. Remove redundant error

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The `run_claim_lock_rewards` function in `contracts/lockdrop/src/execute.rs:632` returns an error if the user does not have a `locked_assets_multiplier`. This case is then checked later in the `user_lock_rewards` function. To simplify the code, the error handling can be removed in `line 632`.

Recommendation

We recommend removing the first instance of the error handling mentioned above.

7. `ura` dependency pointing to latest main git branch

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The `ura` dependency in the root Cargo.toml file points to a git repo. This dependency will always pull the latest version of the main branch in the repo. Although unlikely, breaking changes may be introduced to the `ura-xyz/contracts` repo and therefore savanna contracts as well if the repo is changed before the wasm binary is built.

Recommendation

We recommend either pulling the ura-xyz/contracts folder to store and reference as a local package or referencing a specific commit on the main branch instead of the branch itself to effectively “version” this dependency.

8. Lockdrop account unimplemented query

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The `lockdrop-account` contract does not implement any query. At line `contracts/lockdrop-account/src/contract.rs:43` it's marked with the `unimplemented!` macro.

Recommendation

We recommend implementing at least a config query.

9. Commented code

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The codebase contains multiple instances of commented-out code across various sections:

- `package/src/dex_helper.rs:500`
- `package/src/dex_helper.rs:434`
- `package/src/dex_helper.rs:459`

In the functions stake_lp, unstake_lp, and accrued_rewards, the presence of commented-out code results in these functions being effectively empty .

Recommendation

We recommend removing the unused code.

10. Multiple todo! macro

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The codebase features numerous instances of the todo! macro scattered throughout. The difference between `unimplemented!` and `todo!` is that while `todo!` conveys an intent of implementing the functionality later and the message is “not yet implemented”, `unimplemented!` makes no such claims. Its message is “not implemented”.

- `contracts/lockdrop/src/contract.rs:176`
- `package/src/shared.rs:57`
- `contracts/dex-helper/core/src/contract.rs:208`
- `contracts/dex-helper/core/src/contract.rs:216`
- `contracts/dex-helper/core/src/contract.rs:224`
- `contracts/dex-helper/core/src/contract.rs:232`
- `contracts/dex-helper/core/src/contract.rs:242`
- `contracts/dex-helper/core/src/contract.rs:254`
- `contracts/dex-helper/core/src/contract.rs:262`
- `contracts/dex-helper/core/src/contract.rs:269`
- `test/src/helper.rs:854`

Recommendation

We recommend replacing the todo macros with unimplemented if this matches the intention of the functionality. .

Document Control

Version	Date	Notes
-	26th February 2024	Security audit commencement date.
0.1	5th March 2024	Initial report with identified findings delivered.
0.5	9th - 12th March 2024	Fixes remediations implemented and reviewed.
1.0	13th March 2024	Audit completed, final report delivered.

Appendices

A. Appendix – Risk assessment methodology

SCV-Security employs a risk assessment methodology to evaluate vulnerabilities and identified issues. This approach involves the analysis of both the LIKELIHOOD of a security incident occurring and the potential IMPACT if such an incident were to happen. For each vulnerability, SCV-Security calculates a risk level on a scale of 5 to 1, where 5 denotes the highest likelihood or impact. Consequently, an overall risk level is derived from combining these two factors, resulting in a value from 10 to 1, with 10 signifying the most elevated level of security risk

Risk Level	Range
CRITICAL	10
SEVERE	From 9 to 8
MODERATE	From 7 to 6
LOW	From 5 to 4
INFORMATIONAL	From 3 to 1

LIKELIHOOD and **IMPACT** would be individually assessed based on the below:

Rate	LIKELIHOOD	IMPACT
5	**Extremely Likely**	Could result in severe and irreparable consequences.
4	**Likely**	May lead to substantial impact or loss.
3	**Possible**	Could cause partial impact or loss on a wide scale.
2	**Unlikely**	Might cause temporary disruptions or losses.
1	**Rare**	Could have minimal or negligible impact.

B. Appendix – Report Disclaimer

This report should not be regarded as an "endorsement" or "disapproval" of any specific project or team. These reports do not indicate the economics or value of any "product" or "asset" created by a team or project that engages SCV-Security for a security review. The audit report does not make any statements or warranties about the code's utility, safety, suitability of the business model, regulatory compliance of the business model, or any other claims regarding the fitness of the implementation for its purpose or its bug-free status. The audit documentation is intended for discussion purposes only. The content of this audit report is provided "as is," without representations and warranties of any kind, and SCV-Security disclaims any liability for damages arising from or in connection with this audit report. Copyright of this report remains with SCV-Security.

THANK YOU FOR CHOOSING

SCV SECURITY

scv.services

contact@scv.services