SCV SECURITY

AUDIT REPORT

PFC Team

Auction Pool Contracts

Prepared by SCV-Security

On 27th September 2024

Table of Contents

Table of Contents 2
Introduction 3
Scope Functionality 3
Submitted Codebase 3
Submitted Codebase – Revisions 4
Methodologies 4
Code Criteria 5
Findings Summary 6
Findings Technical Details 7
1. Users could receive rewards for auctions they did not participate in 7
2. Dust token refunds could fail due to out-of-gas error 8
3. Auctions cannot be settled if rewards contain the same denom with remaining_bidding_balance 9
4. Dust token refunds may not work correctly 10
5. Setting fees as 0% or 100% would prevent auctions from being settled 11
6. Code clarity improvements 12
Document Control 13
Appendices 14
A. Appendix - Risk assessment methodology 14
B. Appendix - Report Disclaimer 15

Introduction

SCV has been engaged by Injective Labs to conduct a comprehensive security review with the goal of identifying potential security threats and vulnerabilities within the submitted 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 Injective Auction Pool contract allows community members to pool together INJ to participate in burn auctions. A treasury chest contract is instantiated for won auctions so users can redeem their LP tokens for rewards.

Submitted Codebase

injective-auction-pool	
Repository	https://github.com/PFC-Validator/injective-auction-pool
Commit	`22e546a28a5f8cdbc64f17171a2b1f3630cdcc21`
Contracts	`injective-auction-pool` & `treasurechest-contract`
Branch	`main`

An additional component, specifically the execution bot detailed below, was submitted for further context. However, this component is excluded from the current audit scope.

inj-bot	
Repository	https://github.com/PFC-Validator/inj-bot
Commit	`755e24ed7d9cdf8ffaa9a01ec62a9903891ffe86`
Branch	`main`

Submitted Codebase – Revisions

injective-auction-pool	
Repository	https://github.com/PFC-Validator/injective-auction-pool
Commit	`d4878009ee9ff82de2ab1d8442812ed079f1f57a`
Contracts	`injective-auction-pool` & `treasurechest-contract`
Branch	`fix/scv-audit`

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 PFC. 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**	N/A
Coverage	**SUFFICIENT**	`cargo tarpaulin` reported a coverage of `55.44`%.
Readability	**SUFFICIENT**	N/A
Complexity	**SUFFICIENT**	N/A

Findings Summary

Summary Title	Risk Impact	Status
Users could receive rewards for auctions they did not participate in	CRITICAL	RESOLVED
Dust token refunds could fail due to out-of-gas error	SEVERE	RESOLVED
Auctions cannot be settled if rewards contain the same denom with `remaining_bidding_balance`	MODERATE	RESOLVED
Dust token refunds may not work correctly	MODERATE	RESOLVED
Setting fees as 0% or 100% would prevent auctions from being settled	LOW	RESOLVED
Code clarity improvements	INFO	RESOLVED

Findings Technical Details

1. Users could receive rewards for auctions they did not participate in

RISK IMPACT: CRITICAL	STATUS: RESOLVED

Description

The `join_pool` function in `contracts/injective-auction-pool/src/executions.rs:103` does not validate that the previous auction round is settled before allowing users to join the pool. For example, if the contract wins the auction round, `WHITELISTED_ADDRESSES` must call `SettleAuction` to finalize the auction so the users who participated can receive rewards from the treasury chest contract.

However, if a user joins the pool before the auction is settled via front-running, the user will receive the LP tokens designated for the previous auction round. This allows the user to receive rewards for the auction they did not participate in, which is unfair to other legitimate users who participated in the auction period.

Recommendation

Consider adding validation to ensure the previous auction is settled before allowing users to join the pool. This can be achieved by checking whether the `current_auction_round` equals the `UNSETTLED_AUCTION` round.

2. Dust token refunds could fail due to out-of-gas error

RISK IMPACT: SEVERE	STATUS: RESOLVED

Description

The `return_dust` function in `contracts/treasurechest-contract/src/executions.rs:90-97` iterates over all native tokens in the contract and refunds them to the owner if needed. This is problematic because an attacker can create many token factory denoms and sends them to the contract to cause an out-of-gas error when iterating over the token balances.

As a result, a denial of service issue will occur as the owner cannot withdraw the dust tokens from the contract.

Recommendation

Consider iterating over `TOTAL_REWARDS` token balances instead of all native tokens held by the contract.

3. Auctions cannot be settled if rewards contain the same denom with `remaining_bidding_balance`

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

The `new_auction_round` function in `contracts/injective-auction-pool/src/helpers.rs:112-117` adds the `remaining_bidding_balance` to the `basket_to_treasure_chest` vector if there are unused bidding balances. If the auction basket contains rewards with the same denom as `remaining_bidding_balance` (which is `config.native_denom`), the `basket_to_treasure_chest` vector will include both coins with the same denom in lines 126-129.

This means the contract will try to transfer coins with duplicate denoms in line 169, which will fail because Cosmos SDK does not allow it, preventing auctions from being settled.

Recommendation

Consider modifying the implementation to accumulate the `remaining_bidding_balance` and basket reward amount before sending it as a single coin.

4. Dust token refunds may not work correctly

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

The `return_dust` function in `contracts/treasurechest-contract/src/executions.rs:92-95` refunds the dust token if the ticket’s worth exceeds the contract balance. This is incorrect because the remaining dust token amounts might exceed a ticket's value, which will not be refunded to the owner.

Recommendation

Consider modifying the implementation so the dust tokens are refunded when the contract holds all the tickets mentioned in `contracts/treasurechest-contract/README.md:14`.

For example, if `CONFIG.burn_it` is true, check that the LP token total supply is zero (all tickets are burnt for withdrawals). If `CONFIG.burn_it` is false, check that the contract’s token balance matches the total supply (all tickets are received from withdrawals).

5. Setting fees as 0% or 100% would prevent auctions from being settled

RISK IMPACT: LOW	STATUS: RESOLVED

Description

If the owner configures the reward fee (`packages/injective_auction/src/auction_pool.rs:125`) as 0% or 100%, the `SettleAuction` message will fail to execute because the `basket_fees` or `basket_to_treasure_chest` vector will include zero amount of coin in `contracts/injective-auction-pool/src/helpers.rs:120-130`. Since Cosmos SDK does not allow zero-amount native token transfers, the message will fail, preventing auctions from being settled.

Recommendation

Consider adding validation so the `basket_fees` and `basket_to_treasure_chest` vectors only include coins with amounts larger than zero.

6. Code clarity improvements

RISK IMPACT: INFORMATIONAL	**STATUS: RESOLVED**

Description

The following instances illustrate instances where code readability and maintainability can be improved:

- In `contracts/injective-auction-pool/src/executions.rs:133`, create the `lp_denom` variable earlier and use it in lines 128 and 137.
- In `contracts/injective-auction-pool/src/helpers.rs:192-200` and `contracts/injective-auction-pool/src/helpers.rs:229-237`, the `current_basket` variable defined in line 69 can be directly used.
- `contracts/treasurechest-contract/src/contract.rs:42` can be removed because the `TOTAL_REWARDS` state will be empty during contract instantiation, so clearing it is a redundant operation.
- In `contracts/treasurechest-contract/src/contract.rs:100`, the `CONTRACT_NAME` variable defined in line 22 can be directly used.

Recommendation

Consider applying the recommendations above.

Document Control

Version	Date	Notes
-	6th September 2024	Security audit commencement date.
0.1	18th September 2024	Initial report with identified findings delivered.
0.5	25th September 2024	Fixes remediations implemented and reviewed.
1.0	27th September 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

THANK YOU FOR CHOOSING

SCV
SECURITY

scv.services

contact@scv.services