SCV SECURITY

AUDIT REPORT

Talis

Staking Contract

Prepared by SCV-Security

On 20th March 2024

Table of Contents

Introduction

SCV has been engaged by Talis 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 Talis staking contract enables users to stake xTALIS tokens to earn INJ rewards. The amount of rewards is computed based on the user staked amount proportioned to the total staked amount, along with the total rewards that can be distributed in that epoch.

Submitted Codebase

staking-contract	
Repository	https://github.com/Talis-Art/talis_contracts_v2
Commit	`051ec0aae1fc5175d3de67b9d605038a310ce80d`
Branch	`feat/talis-1387_staking`
Contract	`staking`

Revisions Codebase

staking-contract	
Repository	https://github.com/Talis-Art/talis_contracts_v2
Commit	`7bab1ed66d8c1d1462fc37bf3599588a5e2c37a0`
Branch	`feat/talis-1422_staking_locked_unbonding`
Contract	`staking`

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 Talis. 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**	Detailed documentation is available in the README file and code comments.
Coverage	**SUFFICIENT**	`cw-multi-test` integration tests are implemented in `packages/tge-multitest`.
Readability	**SUFFICIENT**	The codebase had good readability as most functions and state variables are documented thoroughly.
Complexity	**SUFFICIENT**	Reward distribution mechanisms are based on epochs (monthly basis) instead of timestamps or blocks.

Findings Summary

Summary Title	Risk Impact	Status
Inconsistent expired epoch ranges calculations	SEVERE	RESOLVED
Updating the distribution token while the contract has a non-empty balance causes state inconsistency	LOW	RESOLVED
CW20 token is not validated to be the correct denom	LOW	RESOLVED
Admin can transfer unclaimed rewards to any address	LOW	ACKNOWLEDGED
Two-step ownership transfer is not implemented	INFO	RESOLVED
`Instantiate` function emits default attributes	INFO	RESOLVED
Unused error enums	INFO	RESOLVED
`semver` crate is not version locked	INFO	RESOLVED

Findings Technical Details

1. Inconsistent expired epoch ranges calculations

RISK IMPACT: SEVERE	**STATUS: RESOLVED**

Description

When the `garbage_collect_expired` function in `contracts/staking/src/state.rs:198` removes expired rewards from users, the range **excludes** the starting epoch from the active epoch window. For example, if the active epochs are `4..=15` (4 is the start epoch while `15` is the end epoch), `4` is not considered an expired epoch, and the pending rewards remain.

However, when the admin removes expired rewards in the `drain_unclaimed_deposits` function, the pending rewards in the starting epoch (epoch `4`) will be removed. This is because the `expired_distribution_range` function in `contracts/staking/src/reward.rs:269` **includes** the starting epoch from the active epoch window when calculating the expired epoch ranges, causing a discrepancy between the `garbage_collect_expired` and `drain_unclaimed_deposits` functions.

As a result, if the admin calls `drain_unclaimed_deposits` to clear pending rewards for epoch `4` and users withdraw the pending rewards after that, rewards belonging to other epochs will be consumed instead.

Ultimately, the contract will be unable to distribute rewards due to insufficient funds.

Recommendation

Consider modifying the `expired_distribution_range` function not to include the active epochs' start range as an expired epoch.

2. Updating the distribution token while the contract has a non-empty balance causes state inconsistency

RISK IMPACT: LOW	STATUS: RESOLVED

Description

The `distribution_token` in `contracts/staking/src/state.rs:24` records the available funds in the contract for reward distribution. The rewards can be added via the `deposit_reward` function in `contracts/staking/src/contract.rs:186-187`.

Suppose the admin updates the distribution token in line 293 after the contract has distributed funds (e.g., users claim pending rewards or the admin calls the `reimburse` function). In that case, the contract will try to send funds that it does not hold, causing the transaction to fail due to insufficient funds.

The admin can recover this by resetting the distribution token to its previous value, but there would be a shortfall of funds if someone deposited rewards through the `deposit_reward` function.

The other method is for the admin to manually transfer the new distribution token to the contract. But it would cause the previous funds deposited to be stuck and cannot be withdrawn

Recommendation

Consider preventing the distribution token from being modified after the contract is instantiated or only allowing the distribution token to be modified when the contract has no rewards to be withdrawn.

3. CW20 token is not validated to be the correct denom

RISK IMPACT: LOW	STATUS: RESOLVED

Description

The `amount_from_balance` function in `packages/common/src/common.rs:38` does not check whether the provided CW20 token equals the expected `search_denom`. It only checks that the provided token and `search_denom` are both CW20 tokens.

While this will not occur in the audited commit hash because only native tokens can be deposited, it may cause an issue in future development if it is expected to authenticate CW20 tokens correctly.

```
#[test]
fn mismatch_cw20_token() {

    let token1 = "token1";
    let token2 = "token2";

    amount_from_balance(
        Balance::Cw20(
            Cw20CoinVerified {
                address: Addr::unchecked(token1),
                amount: Uint128::new(1000)
            }
        ),
        &CheckedDenom::Cw20(Addr::unchecked(token2))
    ).unwrap();
}
```

Recommendation

Consider validating that the CW20 token is the expected token.

4. Admin can transfer unclaimed rewards to any address

RISK IMPACT: LOW	STATUS: ACKNOWLEDGED

Revision Notes

The team mentions that the risk associated with an admin being malicious or compromised is reduced by setting the admin to the Talis core team's CW3 multisig address. This approach ensures that the activation of the reimbursement feature necessitates a collective decision.

Description

The `reimburse` function in `contracts/staking/src/contract.rs:229` allows the admin to withdraw any unclaimed rewards to any address. This is a dangerous pattern because the caller may specify any target.

In the event of a malicious or compromised admin, the unclaimed rewards in the contract can be sent to any address.

Recommendation

Consider implementing whitelisted addresses that can only be updated via contract migration to ensure unclaimed rewards can only be distributed to approved addresses.

5. Two-step ownership transfer is not implemented

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The codebase does not implement two-step ownership transfer in `contracts/staking/src/contract.rs:285`.

Using a two-step ownership transfer mechanism helps provide a window of opportunity for the current owner to cancel the transfer if they did not intend to initiate it or if there were any unintended actions.

As a result, the ownership will be lost and cannot be recovered if transferred to an incorrect address that no one owns.

Recommendation

Consider implementing a two-step ownership transfer that proposes a new owner in the first step and requires the proposed owner to accept it as the second step.

6. `Instantiate` function emits default attributes

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The `instantiate` function in `contracts/staking/src/contract.rs:68` emits empty default attributes. It is best practice to emit detailed attributes whenever a state change occurs.

Recommendation

Consider emitting detailed attributes in the `instantiate` function response.

7. Unused error enums

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The `EpochAlreadyScheduled`, `DataShouldBeGiven`, `InvalidDenom`, and `DenomNotFound` error enums are implemented in `contracts/staking/src/error.rs` but never explicitly used.

Recommendation

Consider removing all unused error enums to avoid redundant code.

8. `semver` crate is not version locked

RISK IMPACT: INFORMATIONAL	STATUS: RESOLVED

Description

The `semver` crate is only locked to a major version in `contracts/staking/Cargo.toml:51`. It is best practice to lock a minor version and a patch version as well in case any breaking changes are accidentally introduced in any future minor or patch versions.

Recommendation

Consider adding a minor version and a patch version to the `semver` cargo dependency.

Document Control

Version	Date	Notes
-	27th February 2024	Security audit commencement date.
0.1	11th March 2024	Initial report with identified findings delivered.
0.5	18th - 20th March 2024	Fixes remediations implemented and reviewed.
1.0	20th 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