SCV SECURITY

AUDIT REPORT

Dojo Trading Reflection Token

Prepared by SCV-Security

On 20th February 2024

Table of Contents

Table of Contents 2

Introduction 3

Scope Functionality 3

Submitted Codebase 3

Revisions Codebase 4

Methodologies 4

Code Criteria 5

Findings Summary 6

1. ensure_antiwhale can be bypassed 7

2. ensure_antiwhale can potentially block whitelist contracts 7

Findings Technical Details 8

1. Tax Implementation Hinders Swap Functionality 8

2. Taxation Discrepancy in Token Transfers via ExecuteMsg::Send and ExecuteMsg::SendFrom 9

3. Lack of validation on LIQUIDITY_PAIR results in unhandled panic 10

4. Missing validation for both LIQUIDITY_PAIR and REFLECTION_PAIR assets 11

5. reflection_pair[1] is sent as native regardless of its type 12

6. Inconsistent Tax Calculation between query_tax in cw_reflection_token and liquify_treasury in cw20_reflection_treasury 13

7. Wrong increase allowance amount on liquify_treasury 14

8. Lack of address validation 15

9. Improper derive of Treasury address 16

10. Lack of validation for set_tax_rate function 17

11. Non-Conventional Data Return in QueryMsg::QueryRates 18

12. Unnecessary Use of Response Return Type 19

Document Control 20

Appendices 21

A. Appendix - Risk assessment methodology 21

B. Appendix - Report Disclaimer 22

Introduction

SCV has been engaged by Dojo Trading 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 audit evaluates two new contracts within the DojoSwap protocol, a modified CW20 token that incorporates an automated taxation system designed to fund treasury operations and anti-whale measures to ensure market stability and a treasury contract that facilitates liquifying.

Submitted Codebase

Reflection & Treasury Contracts	
Repository	https://github.com/dojo-trading/dojoswap-contracts
Commit	`65c7ca23cd34de7b07273650d960cae2c9a5b79b`
Branch	`main`
Contracts	● `cw20_reflection_token` ● `cw20_reflection_treasury`

Revisions Codebase

Reflection & Treasury Contracts	
Repository	https://github.com/dojo-trading/dojoswap-contracts
Commit	`51a3437613bfbfa3ed968de038208f85d3d7bab1`
Branch	`main`
Contracts	• `cw20_reflection_token` • `cw20_reflection_treasury`

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 Dojo Trading. 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	**NOT-SUFFICIENT**	Coverage tests were not sufficient. The audit submission had poor coverage overall. It's recommended to considerably increase testing coverage around 60% to ensure core components are covered.
Readability	**SUFFICIENT**	The codebase had good readability overall; however, it fell short in adhering to Rust and CosmWasm best practices due to an abundance of unwrap usage.
Complexity	**SUFFICIENT**	N/A

Findings Summary

Summary Title	Risk Impact	Status
Tax Implementation Hinders Swap Functionality	**SEVERE**	**RESOLVED**
Taxation Discrepancy in Token Transfers via `ExecuteMsg::Send` and `ExecuteMsg::SendFrom`	**SEVERE**	**RESOLVED**
Lack of validation on `LIQUIDITY_PAIR` results in unhandled panic	**SEVERE**	**ACKNOWLEDGED**
Missing validation for both `LIQUIDITY_PAIR` and `REFLECTION_PAIR` assets	**SEVERE**	**PARTIALLY RESOLVED**
`reflection_pair[1]` is sent as native regardless of its type	**SEVERE**	**RESOLVED**
Inconsistent Tax Calculation between `query_tax` in `cw_reflection_token` and `liquify_treasury` in `cw20_reflection_treasury`	**MODERATE**	**RESOLVED**
Wrong increase allowance amount on `liquify_treasury`	**LOW**	**RESOLVED**
Lack of address validation	**LOW**	**RESOLVED**
Improper derive of Treasury address	**LOW**	**ACKNOWLEDGED**
Lack of validation for `set_tax_rate` function	**LOW**	**RESOLVED**
Non-Conventional Data Return in `QueryMsg::QueryRates`	**LOW**	**ACKNOWLEDGED**
Unnecessary Use of Response Return Type	**INFO**	**ACKNOWLEDGED**

Audit Observations

The audit observations section is intended to present potential findings that are related to the underlying design of the protocol and would require underlying design changes to remediate that may change the overall functioning of the protocol. SCV asks that the client formulate responses to add context to validate or invalidate the following concerns.

1. `ensure_antiwhale` can be bypassed

The anti-whale mechanism in `contracts/cw20_reflection_token/src/contract.rs:634` is designed to mitigate market volatility by restricting large token disposals. However, the current implementation evaluates transactions individually against predefined thresholds, neglecting the cumulative effect of multiple transactions within a single block. This oversight allows a whale to bypass the restrictions by dividing a large transaction into numerous smaller ones executed in quick succession. Such a strategy can still significantly influence the market, defeating the intent of stabilising the token's price and market.

2. `ensure_antiwhale` can potentially block whitelist contracts

The `ensure_antiwhale` function is unintentionally affecting operations and contracts that are meant to be exempt from its constraints. For example, it restricts users from adding liquidity to a pool when the amount they wish to supply surpasses the established anti-whale threshold. Critical contracts, such as the treasury contract, may need to handle amounts that exceed this threshold, resulting in situations where the token's functionality is permanently hindered.

Refining the ensure_antiwhale function is essential to accommodate legitimate operations and contract interactions, ensuring the token's functionality is preserved and remains seamless for all necessary transactions.

Findings Technical Details

1. Tax Implementation Hinders Swap Functionality

RISK IMPACT: SEVERE	STATUS: RESOLVED

Description

During liquidity provision via the `execute_transfer_from` method in `contracts/cw20_reflection_token/src/contract.rs:299`, the pool expects to transfer a specific amount X from a user's balance. However, it receives X minus the fee, while the minting of LP tokens still relies on the original amount X.

This results in the minting of LP tokens based on the pre-fee amount rather than the post-fee amount. In particular, when a contract executes a `transfer_from`, if it is unaware that the CW20 token it wishes to transfer applies fees, it will receive fewer tokens than expected.

Recommendation

We recommend implementing a whitelist of addresses exempt from the tax, specifically for transactions that use the `execute_transfer_from` method.

2. Taxation Discrepancy in Token Transfers via `ExecuteMsg::Send` and `ExecuteMsg::SendFrom`

RISK IMPACT: SEVERE	**STATUS: RESOLVED**

Description

During the execution of `ExecuteMsg::Send` and `ExecuteMsg::SendFrom` in `contracts/cw20_reflection_token/src/contract.rs:219` and `370` respectively, it is found that the token amount specified in the `Cw20RceiveMsg`, sent to the recipient contract, does not include taxes. This leads to issues, especially in liquidity pools, where the expected and actual received token amounts differ due to taxation. This discrepancy can affect the accurate tracking of token flows and impact operations that rely on precise token valuations.

Recommendation

We recommend replacing the `amount` in `contracts/cw20_reflection_token/src/contract.rs:290` and `444` with `outgoing_amount`.

3. Lack of validation on `LIQUIDITY_PAIR` results in unhandled panic

RISK IMPACT: SEVERE	STATUS: ACKNOWLEDGED

Revision Notes

The team advises that this behaviour is intended. It is expected to trigger an error upon the `liquify_treasury` function being triggered should the pair contract not be specified in the transaction.

Description

In the `liquify_treasury` function in `contracts/cw20_reflection_treasury/src/contract.rs:132-135`, the contract attempts to read liquidity pools and assets. Should these values not have been previously recorded, the application of unwrap triggers a panic, thereby causing the transaction to fail in its entirety.

Recommendation

We recommend adding validation checks to ensure the presence of necessary data before accessing it. When the data is not found, the contract should return a default response instead.

4. Missing validation for both `LIQUIDITY_PAIR` and `REFLECTION_PAIR` assets

RISK IMPACT: SEVERE	**STATUS: PARTIALLY RESOLVED**

Revision Notes

The team partially implemented SCV's recommendation. The contract might panic in particular edge cases forcing the transaction to be reverted. Team advises as intended and expected behaviour.

Description

In the `set_liquidity_pair` and `set_reflection_pair` functions in `contracts/cw20_reflection_treasury/src/contract.rs:315` and `331` respectively, the contract stores specific assets for the pools without first verifying that the pools contain these assets.

Additionally, the code assumes the positions of assets within the pools without conducting any form of validation. It assumes that `liquidity_pair[0]` represents the CW20 token that initializes the treasury contract, `liquidity_pair[1]` and `reflection_pair[1]` correspond to INJ tokens, and `reflection_pair[0]` is identified as a DOJO token.

Recommendation

We recommend verifying that the contract is indeed a liquidity pool and confirming the specified assets are present. If assets are structured as lists, verifying their order is also necessary.

5. `reflection_pair[1]` is sent as native regardless of its type

RISK IMPACT: SEVERE	STATUS: RESOLVED

Description

The `token_denom` variable in `contracts/cw20_reflection_treasury/src/contract.rs:174` of the `liquify_treasury` function serves to identify the denomination or address of `reflection_pair[1]`, which is utilized as the token for liquidity provision. However, this is consistently sent as a native token in the liquidity provision message, regardless of whether it is a CW20 token or a native token.

Recommendation

We recommend that when `reflection_pair[1]` is a native token, it should be sent as a coin in the transaction. Conversely, if `reflection_pair[1]` could potentially be a CW20 token, an increase allowance message should precede the liquidity provision message.

6. Inconsistent Tax Calculation between `query_tax` in `cw_reflection_token` and `liquify_treasury` in `cw20_reflection_treasury`

RISK IMPACT: MODERATE	STATUS: RESOLVED

Description

There is an inconsistency in the way taxes are calculated between the `query_tax` function in `contracts/cw20_reflection_token/src/contract.rs:553` and the `liquify_treasury` function in `contracts/cw20_reflection_treasury/src/contract.rs:120`.

In the `query_tax` function, the calculation is as follows:

1. `reflection_amount` = `taxed_amount` * `reflection_rate`
2. `liquidity_amount` = `taxed_amount` - `reflection_amount`
3. `burn_amount` = 0

However, in `liquify_treasury`, the process involves querying the `cw20_reflect_token` for `reflection_rate` and `burn_rate`, leading to a different computation:

1. `taxed_amount` = the total balance of the treasury
2. `reflection_amount` = `taxed_amount` * `reflection_rate`
3. `burn_amount` = `taxed_amount` * `burn_rate`
4. `liquidity_amount` = `taxed_amount` - `reflection_amount` - `burn_amount`

Recommendation

We recommend matching the calculation on `query_tax` to reflect the calculation on `liquify_treasury`.

7. Wrong increase allowance amount on `liquify_treasury`

RISK IMPACT: LOW	STATUS: RESOLVED

Description

During the execution of the `liquify_treasury` function, when `liquidity_amt` exceeds zero, half of the asset is converted into INJ before being contributed to liquidity. However, the increased allowance is erroneously set to the total available amount rather than just the unswapped portion.

Recommendation

We recommend modifying the increase allowance specifically to the difference between `liquidity_amt` and `swap_amount`.

8. Lack of address validation

RISK IMPACT: LOW	STATUS: RESOLVED

Description

Throughout the codebase, there are several instances where there is an absence of necessary address validations. The instances include:

- `contracts/cw20_reflection_token/src/contract.rs:617`
- `contracts/cw20_reflection_token/src/contract.rs:56`
- `contracts/cw20_reflection_treasury/src/contract.rs:46`

Recommendation

We recommend validating the addresses where specified.

9. Improper derive of Treasury address

RISK IMPACT: LOW	STATUS: ACKNOWLEDGED

Description

In the `register_deployment` function in contracts/cw20_reflection_token/src/contract.rs:695, the current approach of obtaining the treasury's address through reading events is neither the most efficient nor the most secure method available.

Recommendation

We recommend using a more robust solution involving deriving the contract address using `cw0::parse_instantiate_response_data(data)`.

10. Lack of validation for `set_tax_rate` function

RISK IMPACT: LOW	STATUS: RESOLVED

Description

The `set_tax_rate` function in `contracts/cw20_reflection_token/src/contract.rs:590` lacks essential validation for tax-related parameters, which can result in inconsistencies and errors within tax rate configurations that could disrupt the token.

Recommendation

We recommend introducing the following validation checks for all rate updates to ensure they are within acceptable ranges:

- `TAX_RATE` must be less than or equal to 1
- The sum of `REFLECTION_RATE` + `BURN_RATE` must be less than or equal to 1
- `MAX_TRANSFER_SUPPLY_RATE` should not exceed 1

11. Non-Conventional Data Return in `QueryMsg::QueryRates`

RISK IMPACT: LOW	STATUS: ACKNOWLEDGED

Description

The `query_rate` function in `contracts/cw20_reflection_token/src/contract.rs:571` currents returns a tuple containing four `Decimal` values. It is generally advised against returning a tuple of this magnitude due to concerns over diminished clarity and the increased complexity in understanding and managing the data.

Recommendation

We recommend establishing a dedicated `Response` struct with named fields for each tax value.

12. Unnecessary Use of Response Return Type

RISK IMPACT: INFO	STATUS: ACKNOWLEDGED

Description

The `ensure_admin` and `ensure_antiwhale` functions located in `contracts/cw20_reflection_token/src/contract.rs:622` and `634` respectively, currently return a `Result<Response, ContractError>`. However, the `Response` value returned by these functions is not utilised in subsequent operations. Instead, operations only leverage the error-checking component of the result.

Recommendation

We recommend replacing the return type from `Result<Response, ContractError>` with `Result<(), ContractError>`.

Document Control

Version	Date	Notes
-	5th February 2024	Security audit commencement date.
0.1	13th February 2024	Initial report with identified findings delivered.
0.5	19th February 2024	Fixes remediations implemented and reviewed.
1.0	20th February 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