AUDIT REPORT

DoJo Trading
susdi token Contract

Prepared by SCV-Security

On 6th June 2024

Table of Contents

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 staking contract allows users to deposit USDI tokens in exchange for sUSDI tokens, a staked version of USDI. sUSDI tokens will increase in value over time as USDI rewards are accrued to the vault. When redeeming sUSDI for USDI, users must wait a 7-day unbonding period before claiming the underlying USDI.

Submitted Codebase

susdi-token	
Repository	https://github.com/dojo-trading/dojoswap-contracts
Commit	245fccdab163f87ae45d7fdc84c1f635e5535483
Branch	main
Contract	susdi_token

Revisions Codebase

susdi-token	
Repository	https://github.com/dojo-trading/dojoswap-contracts
Commit	baec3f912c5feef57542d0d6e5d948d33080a15a
Branch	main
Contract	susdi_token

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**	There are no test cases in the codebase.
Readability	**SUFFICIENT**	N/A
Complexity	**SUFFICIENT**	N/A

Findings Summary

Summary Title	Risk Impact	Status
Incorrect staking contract implementation	-	**RESOLVED**
Unused parameter when instantiating contract	**INFO**	**ACKNOWLEDGED**
Storage state is not placed in the correct file	**INFO**	**ACKNOWLEDGED**

Findings Technical Details

1. Incorrect staking contract implementation

RISK IMPACT: -	**STATUS: RESOLVED**

Description

The staking contract's intended functionality allows users to deposit USDI tokens in exchange for sUSDI tokens and vice versa. However, a few issues prevent the contract from functioning properly.

1. The `execute_deposit` function in `contracts/susdi_token/src/contract.rs:106` accepts the USDI tokens from the user and mints sUSDI (shares) to the user.

   The issue is that the `query_deposit_amount` function in `contracts/susdi_token/src/contract.rs:282` computes the amount of the shares to mint without checking the total supply is not zero. This is needed because when the first user deposits USDI to the contract, the total supply is zero, resulting in zero shares and causing the transaction to revert in https://github.com/apollodao/cw-vault-token/blob/9f139f71ea3c94845ef78963fd5651fea9d2c098/src/implementations/cw4626.rs#L89-L91.

   As a result, users cannot deposit funds and mint shares from the contract.

2. The `query_deposit_amount` function in `contracts/susdi_token/src/contract.rs:282-291` uses the contract's USDI balance as the total deposits when computing the amount of sUSDI to mint. This is incorrect due to the following reasons:

   a. The total deposit amount should be subtracted from the user deposit amount as the contract balance already includes it.

   b. The total deposit amount should also be subtracted from the total claims amount, as sUSDI redemption requires a 7-day unstaking period before releasing USDI tokens. This issue also affects the `query_redeem_amount` function in `contracts/susdi_token/src/contract.rs:310-315`.

   As a result, users will receive an incorrect number of shares and tokens.

3. The `Cw20HookMsg::Redeem` message in `contracts/susdi_token/src/contract.rs:62` allows users to redeem their sUSDI tokens for USDI tokens.

   The issue is that it checks the caller as the contract address, which is incorrect as the sUSDI tokens are recorded in the `BALANCES` state. This is also the same issue in `contracts/susdi_token/src/contract.rs:161-166`.

   As a result, users cannot redeem their sUSDI tokens for USDI tokens.

4. The logic for redeeming sUSDI tokens for USDI tokens in `contracts/susdi_token/src/contract.rs:169-173` is incorrect. As the `burn` function from CW4626 reduces the total supply first in https://github.com/apollodao/cw-vault-token/blob/9f139f71ea3c94845ef78963fd5651fea9d2c098/src/implementations/cw4626.rs#L133-L136, the total supply used in `contracts/susdi_token/src/contract.rs:307` will be incorrect as it already been reduced.

   As a result, users will receive an incorrect number of shares and tokens, and the last user cannot unstake their sUSDI completely.

5. The staking contract does not implement protection against share inflation attacks. This means that an attacker can be the first depositor of the staking contract to steal funds from the subsequent depositors.

Recommendation

Consider applying the following remediations:

1. Create a storage state that records the total claims amount when creating new claims in `contracts/susdi_token/src/contract.rs:184`. This state should increase whenever `create_claim` is called and decrease whenever `claim_tokens` is called in `contracts/susdi_token/src/contract.rs:207-219`.

   When the `query_deposit_amount` and `query_redeem_amount` functions are called, the contract balance should be deducted from the total claims amount to prevent funds belonging to other users from being computed as shares.

2. In `execute_redeem`, remove the validation for checking sent funds in `contracts/susdi_token/src/contract.rs:159-166` and implement `cw4626.receive` so the caller's funds are transferred to the contract balance.

   This is required because the `burn` function will decrease the balance from the current contract address, as shown in https://github.com/apollodao/cw-vault-token/blob/9f139f71ea3c94845ef78963fd5651fea9d2c098/src/implementations/cw4626.rs#L231-L245 and https://github.com/apollodao/cw-vault-token/blob/9f139f71ea3c94845ef78963fd5651fea9d2c098/src/implementations/cw4626.rs#L125-L131.

   This is paired with the next recommendation.

3. Remove the `VaultExecuteMsg::Receive` message in `contracts/susdi_token/src/contract.rs:53` and modify the `VaultStandardExecuteMsg::Redeem` message in `contracts/susdi_token/src/contract.rs:92` directly to call the `execute_redeem` function in `contracts/susdi_token/src/contract.rs:144`. This will be the entry point for users to redeem their sUSDI tokens for USDI tokens.

4. In `execute_redeem`, switch the ordering for the CW4626 `burn` function and `query_redeem_amount` function in

contracts/susdi_token/src/contract.rs:169 and contracts/susdi_token/src/contract.rs:172. The query_redeem_amount function must be called before the burn function to ensure the total supply is only reduced after computing the USDI return amount.

5. The query_deposit_amount function in contracts/susdi_token/src/contract.rs:288-291 should deduct the contract balance (underlying_balance variable) with the user deposit amount (amount variable) to prevent double calculation of the deposited funds. Note that this will also be paired to deduct the total claims amount.

6. The query_deposit_amount function in contracts/susdi_token/src/contract.rs:293 should check whether the total shares are zero. If the total shares are zero, compute the shares amount equal to the user deposit amount. If the total shares are not zero, compute the share amount according to the current formula.

7. In execute_redeem, modify contracts/susdi_token/src/contract.rs:186 to use the recipient address instead of info.sender to support sUSDI redemption for other users.

8. Modify the query_total_assets function in contracts/susdi_token/src/contract.rs:324 to exclude the total claims amount when querying the vault's total assets.

9. Remove contracts/susdi_token/src/contract.rs:317-319 as total shares would never be zero when redeeming sUSDI tokens.

10. Implement virtual shares and decimal offsets when computing the number of shares to defend against the first depositor attack, as explained in https://blog.openzeppelin.com/a-novel-defense-against-erc4626-inflation-attacks.

11. Add extensive test cases to ensure the contract works properly. There are no test cases in the codebase.

2. Unused parameter when instantiating contract

RISK IMPACT: INFORMATIONAL	**STATUS:** ACKNOWLEDGED

Description

The `admin` field in `contracts/susdi_token/src/msg.rs:11` is not used when instantiating the contract.

Recommendation

Consider removing it from the `InstantiateMsg` struct to increase code readability.

3. Storage state is not placed in the correct file

RISK IMPACT: INFORMATIONAL	STATUS: ACKNOWLEDGED

Description

The `MAIN_TOKEN` state in `contracts/susdi_token/src/contract.rs:28` is not placed in the `contracts/susdi_token/src/state.rs` file.

Recommendation

Consider moving the `MAIN_TOKEN` state to the `contracts/susdi_token/src/state.rs` file with other storage states to increase code readability and maintainability.

Document Control

Version	Date	Notes
-	8th May 2024	Security audit commencement date.
0.1	14th May 2024	Initial report with identified findings delivered.
0.5	27th May 2024	Fixes remediations implemented and reviewed.
1.0	6th June 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