PUBLIC

Code Assessment

of the Savings Dai Smart Contracts

May 8, 2023

Produced for

by

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help Oazo Apps Limited with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Savings Dai according to Scope to support you in forming an opinion on their security risks.

Savings DAI implements a tokenized EIP 4626 compliant wrapper for DAI Savings Rate. This latest iteration of the code adds a referral feature.

The most critical subjects covered in our audit are functional correctness, security of the assets and adherence to the EIP standards. General subjects covered are optimizations and usability.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the `src` folder of the Savings Dai repository based on the documentation files. The following files from the repository `src` folder were part of the assessment scope:

```
SavingsDai.sol
```

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	12 December 2022	240fe759b00f2effdd7e0feeabc77d258de0ba6e	Initial Version
2	12 January 2023	a773f18328f4a21e3a792975a48393b9d04f7afc	Gas Optimization
3	5 May 2023	9a993bde3fa3ac8cd80549fbafa9a229ba1cca8f	Referral Code Feature

For the solidity smart contracts, the compiler version `0.8.17` was chosen.

2.1.1 Excluded from scope

Any contracts not mentioned above and testing contracts might rely on scope contracts are not part of the scope. Imported libraries and interacting contracts are assumed to behave according to their specification and are not part of the assessment scope.

3 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Oazo Apps Limited offers a contract for users to deposit DAI and earn rewards. The `SavingsDai` interacts with the `Vat` and the `Pot` to track deposited amounts and compute the rewards. The reward grows with time at a rate that can be set by the Oazo Apps Limited governance in the `Pot` contract. The rewards computation and distribution is managed by the `Pot`, but in order to track the shares of each user, the contract is implemented as a tokenized vault and implements EIP 4626.

3.1 Contracts

SavingsDai

The contract implements an tokenized (ERC20 compliant, 18 decimals, "Savings Dai", "sDAI") wrapper around the Pot (DAI Saving Rate). Additionally the contract also implements the functions `increaseAllowance` and `decreaseAllowance`. `SavingsDai` extends the ERC20 features with `permit` and `ERC1271` for the signature validity, which allows contracts to act as they "signed" permits. For improved user experience the contract adheres to the EIP712 standard for the permits. Permits issued by a signer can only be used in order of their nonce.

In order to ease DAI savings for users and track the rewards distribution, the contract tracks the shares each user has by extending the ERC20 with EIP 4626. The shares are computed as normalized amounts with the formula `depositedDaiAmount * RAY / pot.chi()`, where `pot.chi()` is the always increasing rate accumulator. The underlying asset managed by the vault is DAI. The state changing functions related to EIP 4626 are:

- `deposit`: users can deposit a given amount of DAI and can also specify a recipient. The function will first trigger `pot.drip()` to update `pot.chi` if needed. The amount to be deposited is then transferred from the caller and joined with `daiJoin.join`, the corresponding number of shares are also joined at the `Pot` contract. Finally, the shares amount is minted to the receiver.
- `mint`: users can specify a target number of shares and a recipient. The function will first trigger `pot.drip()` to update `pot.chi` if needed. The corresponding DAI amount is computed with `shares * pot.chi() / RAY`. The amount to be deposited is then transferred from the caller and joined with `daiJoin.join`, the corresponding number of shares are also joined at the `pot`. Finally, the shares amount is minted to the receiver.
- `withdraw`: users can specify the amount of DAI they want to be exited from the system, as well as a receiver and an owner. The function will first trigger `pot.drip()` to update `pot.chi` if needed. Allowance is decreased by the corresponding amount of shares if `msg.sender != owner`, then the shares are burned, the same amount of shares are exited from the `Pot` contract and the receiver gets the desired amount of DAI.
- `redeem`: users can specify the amount of shares they want to redeem, as well as a receiver and an owner. The function will first trigger `pot.drip()` to update `pot.chi` if needed. The corresponding DAI amount is computed with `shares * pot.chi() / RAY`. Allowance is decreased by the amount of shares if `msg.sender != owner`, then the shares are burned, the same amount of shares are exited from the `Pot` contract and the receiver gets the corresponding amount of DAI.

Changes in Version 3

Additional functions for `deposit()` and `mint()` have been added which take a referral code as extra parameter. These functions emit a referral event.

3.2 Trust model

Users: untrusted

Contracts used by `SavingsDai`, i.e., `Vat, DaiJoin, Dai`, and `Pot` : fully trusted

4 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

6 Findings

In this section, we describe our findings. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Gas Optimizations

Informational **Version 1**

ISSUEIDPREFIX-001

1. In the `_mint` function, the update of `totalSupply` can be done in the `unchecked` block to save gas. The total supply of shares is bound to be <= total supply of DAI, which is bound to `type(uin256).max`.
2. The internal function `_rpow` always take RAY as `base`, replacing `base` by RAY in the code will save a bit of gas at runtime.

Code corrected:

1. The update of `totalSupply` has been moved in the `unchecked` block.
2. The `base` parameter of the function `_rpow` has been removed, and replaced by RAY everywhere.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Known Attack Vector on `approve()`

Note Version 1

Users should be aware of the well known attack vector on `approve()` (front running changes to existing approvals, spending more tokens than intended by the owner). If needed, they should use the provided `increaseAllowance()` / `decreaseAllowance()` to mitigate this risk.