MANSA

MANSA SMART CONTRACTS SECURITY AUDIT REPORT

OXORIO

JUNE 5, 2024

CONTENTS

1. AUDIT OVERVIEW 4

1.1. PROJECT BRIEF 5

1.2. PROJECT TIMELINE 6

1.3. AUDITED FILES 7

1.4. PROJECT OVERVIEW 8

1.5. CODEBASE QUALITY ASSESSMENT 9

1.6. SUMMARY OF FINDINGS 11

1.7. CONCLUSION 13

2. FINDINGS REPORT 14

2.1. CRITICAL 15

C-01 Comparison of two variables with different decimals in Mansa 15

C-02 Possible irrevocable deletion of a tranche along with user funds in Mansa 17

C-03 Admin loses ability to withdraw excessUsdc after user redemptions in Mansa 18

2.2. MAJOR 19

M-01 Insufficient state transition checks for tranches in Mansa 19

M-02 Missing validation for custodianAddress_ in Mansa 21

2.3. WARNING 22

W-01 Missing validation for redeemRatioBip_ in Mansa 22

W-02 "Dust" on balance after redemption blocks tranche deletion in Mansa 24

W-03 Possible redemption of zero USDC in Mansa 26

W-04 Funds of a removed user from the whitelist are locked in the contract in Whitelist 28

2.4. INFO 29

I-01 Absence of whitelist allows injection and distribution of "dirty" cryptocurrency in Mansa 29

I-02 Calculation of the difference in decimals can be moved to a constant in Mansa 31

I-03 Magic number in Mansa 32

I-04 State Invalid is not used in Mansa.sol 33
I-05 Floating pragma 34
I-06 Usage of string instead of uint128 for storing UUID in Mansa 35
I-07 Unused duration field in Mansa 36
3. APPENDIX 37
3.1. DISCLAIMER 38
3.2. SECURITY ASSESSMENT METHODOLOGY 39
3.3. CODEBASE QUALITY ASSESSMENT REFERENCE 41
Rating Criteria 42
3.4. FINDINGS CLASSIFICATION REFERENCE 43
Severity Level Reference 43
Status Level Reference 43
3.5. ABOUT OXORIO 45

1 AUDIT OVERVIEW

1.1 PROJECT BRIEF

Title	Description
Client	Mansa
Project name	Mansa
Category	Decentralized Finance
Website	https://www.mansafinance.co/
Repository	https://github.com/mansafinance/mansa-contracts
Documentation	-
Initial Commit	d375a4ba61042687fd7264091d550605dbef655d
Final Commit	4005d0db4024befd8ad9994ae05f4721367f92ae
Platform	L2
Network	Base / Polygon / Arbitrum
Languages	Solidity
Lead Auditor	Artem Belozerov - artem@oxor.io
Project Manager	Viktor Mikhailov - viktor@oxor.io

1.2 PROJECT TIMELINE

The key events and milestones of the project are outlined below.

Date	Event
May 24, 2024	Client approached Oxorio requesting an audit.
May 27, 2024	The audit team commenced work on the project.
May 31, 2024	Submission of the comprehensive report.
June 3, 2024	Client feedback on the report was received.
June 5, 2024	Submission of the final report incorporating client's verified fixes.

1.3 AUDITED FILES

The following table contains a list of the audited files. The scc tool was used to count the number of lines and assess complexity of the files.

	File	Lines	Blanks	Comments	Code	Complexity
1	contracts/DummyWhitelist.sol	11	2	1	**8**	0
2	contracts/Mansa.sol	146	16	2	**128**	12
3	contracts/MansaTrancheToken.sol	36	6	1	**29**	14
4	contracts/TestToken.sol	16	4	1	**11**	0
5	contracts/Whitelist.sol	20	4	1	**15**	13
	Total	**229**	**32**	**6**	**191**	**11**

Lines: The total number of lines in each file. This provides a quick overview of the file size and its contents.

Blanks: The count of blank lines in the file.

Comments: This column shows the number of lines that are comments.

Code: The count of lines that actually contain executable code. This metric is essential for understanding how much of the file is dedicated to operational elements rather than comments or whitespace.

Complexity: This column shows the file complexity per line of code. It is calculated by dividing the file's total complexity (an approximation of cyclomatic complexity that estimates logical depth and decision points like loops and conditional branches) by the number of executable lines of code. A higher value suggests greater complexity per line, indicating areas with concentrated logic.

1.4 PROJECT OVERVIEW

The project allows users to invest funds, which are then transferred to a custodian who generates profit.

In the first stage, users invest their funds into a specific tranche, receiving `Mansa` tokens in return. These funds are subsequently transferred to the custodian. Later, the funds, along with the generated profit, are returned to the protocol, allowing users to redeem the initial funds and the accrued profit for their `Mansa` tokens.

1.5 CODEBASE QUALITY ASSESSMENT

The Codebase Quality Assessment table offers a comprehensive assessment of various code metrics, as evaluated by our team during the audit, to gauge the overall quality and maturity of the project's codebase. By evaluating factors such as complexity, documentation and testing coverage to best practices, this table highlights areas where the project excels and identifies potential improvement opportunities. Each metric receives an individual rating, offering a clear snapshot of the project's current state, guiding prioritization for refactoring efforts, and providing insights into its maintainability, security, and scalability. For a detailed description of the categories and ratings, see the Codebase Quality Assessment Reference section.

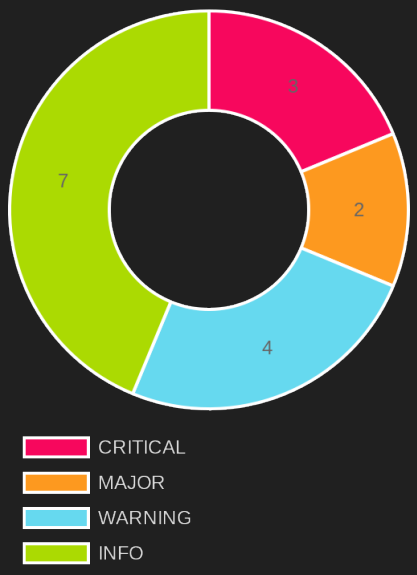
Category	Assessment	Result
Access Control	The project implements a clear and simple access control mechanism. However, specific concerns outlined in `I-01` warrant further review to reinforce the robustness of these operations.	**Good**
Arithmetic	The project has several issues with mathematical operations, particularly those related to rounding during division and working with tokens of different decimals. It is crucial to address these shortcomings meticulously to enhance the overall reliability of the system.	**Poor**
Complexity	The project structure is simple and comprehensible, which positively impacts code readability and ease of work.	**Excellent**
Data Validation	The project performs data validation across many components, but a significant portion of the issues highlighted in this report stem from insufficient validation processes. It is crucial to enhance the validation mechanisms to address these deficiencies and improve the overall robustness of the system.	**Poor**
Decentralization	The project does not incorporate a decentralized approach to management, and therefore, the metric is not applicable in this context.	**Not Applicable**
Documentation	The project lacks any form of documentation, and comments in the codebase are virtually non-existent. The absence of a description of the project's architecture and its functional components hinders the quick understanding of the overall system structure and its operational sequence.	**Absent**

Category	Assessment	Result
External Dependencies	The project lacks significant external dependencies, except for the transfer of invested funds to the custodian's address, which could be a critical point if the custodian is compromised. However, the custodian's operations are beyond the scope of this audit. Nonetheless, to enhance the overall reliability from the protocol's side, greater care should be taken in the custodian's address validation process, as mentioned in the report in `M-02`.	**Not Applicable**
Error Handling	The project demonstrates competent exception handling throughout the codebase. However, it is important to address the issues outlined in the report that highlight potential error scenarios.	**Excellent**
Logging and Monitoring	The project involves token transfers that emit events, providing an opportunity for logging. However, transfer events alone are insufficient for a comprehensive logging system that would effectively utilize third-party monitoring services. Adding alerts for key events within the system will facilitate real-time data analysis and enhance the ability to accurately track system performance and security incidents.	**Poor**
Low-Level Calls	The project is free from low-level calls, ensuring a higher level of security by avoiding potential pitfalls associated with direct, low-level interactions with the blockchain.	**Not Applicable**
Testing and Verification	The project includes tests that verify core functionality with valid data. However, not all tests are current and operational out of the box. Additionally, the low number of tests and limited code coverage create noticeable gaps in testing, especially in critical scenarios that remain untested, leading to issues described in the report. Addressing these gaps will improve the reliability of the testing environment and ensure more comprehensive verification of system behavior under various conditions.	**Poor**

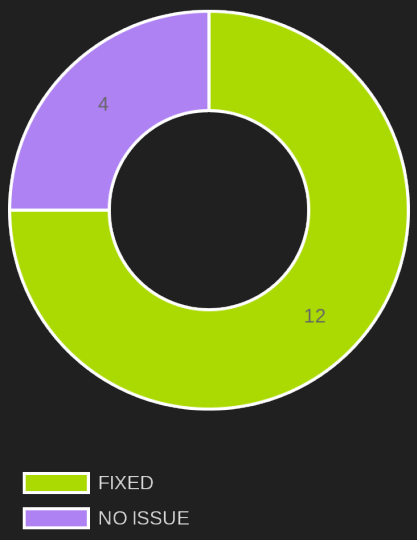
1.6 SUMMARY OF FINDINGS

The table below provides a comprehensive summary of the audit findings, categorizing each by status and severity level. For a detailed description of the severity levels and statuses of findings, see the Findings Classification Reference section.

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**3**	0	3	0	0
MAJOR	**2**	0	2	0	0
WARNING	**4**	0	3	0	1
INFO	**7**	0	4	0	3
TOTAL	**16**	**0**	**12**	**0**	**4**



Issue distribution by severity



Issue distribution by status

This table provides an overview of the findings across the audited files, categorized by severity level. The table enables to quickly identify areas that require immediate attention and prioritize remediation efforts accordingly.

File	TOTAL	CRITICAL	MAJOR	WARNING	INFO
contracts/Mansa.sol	**14**	3	2	3	6
contracts/Whitelist.sol	**1**	0	0	1	0

1.7 CONCLUSION

Overall, the project architecture is simple and understandable, which positively impacts the perception of the codebase. However, the project has problematic areas, particularly in arithmetic operations, insufficient data validation, and edge case handling. We advise thoroughly analyzing and addressing the issues described below to enhance the reliability and security of the system. By resolving these identified problems, the project can significantly improve its resilience to potential vulnerabilities and ensure a more robust operational structure.

2 FINDINGS REPORT

OXORIO

2.1 CRITICAL

C-01	Comparison of two variables with different decimals in `Mansa`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `transitionTrancheState`	112

Description

In the function `transitionTrancheState` of the `Mansa` contract, the comparison occurs between the USDC balance - `tranche.usdcBalance`, and the maximum redeemable USDC amount - `maxRedeemableUsdcAmount`. Here, `tranche.usdcBalance` is stored with `6` decimals, whereas `maxRedeemableUsdcAmount` is calculated with `18` decimals, since `tranche.redeemableTokenSupply` has `18` decimals:

```
tranche.redeemableTokenSupply = tranche.token.totalSupply();
uint256 maxRedeemableUsdcAmount = tranche.redeemableTokenSupply * tranche.redeemRatioBip /
10000;
if (tranche.usdcBalance > maxRedeemableUsdcAmount) {
    // ...
```

This causes the `if` block condition to work incorrectly, as `tranche.usdcBalance` equal to `150` USDC with `6` decimals will be less than `maxRedeemableUsdcAmount` equal to `110` USDC with `18` decimals.

Recommendation

We recommend comparing the current balance and the maximum redeemable amount by converting them to the same decimals.

Update

Fixed in commit 5bc426b32a85598a8e46be79cc4c9e72e6049b6d

C-02	Possible irrevocable deletion of a tranche along with user funds in `Mansa`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `createTranche`	56

Description

In the function `createTranche` of the `Mansa` contract, when creating a tranche, the admin maps the `uuid` to the new tranche. There is no check for the existence of a tranche with the given `uuid`.

This allows the admin to accidentally or intentionally overwrite an existing tranche. In such a case, access to the tranche would be lost along with the users' invested funds.

Recommendation

We recommend adding more checks when creating a tranche to prevent overwriting an existing tranche with funds.

Update

Fixed in commit 5bc426b32a85598a8e46be79cc4c9e72e6049b6d

Mansa's response:

This is not a vulnerability and rather a suggestion to prevent careless mistakes as the function is only callable by contract admins. It only becomes an issue if admin carelessly deletes the tranche with funds in it. Nevertheless this concern has been addressed in `5bc426b32a85598a8e46be79cc4c9e72e6049b6d` by checking for the inexistence of a tranche before creation.

C-03	Admin loses ability to withdraw `excessUsdc` after user redemptions in `Mansa`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `withdrawExcessUSDC`	88

Description

In the function `withdrawExcessUSDC` of the `Mansa` contract, excess USDC tokens are withdrawn, which exceed the amount of tokens designated for redemption. However, if users start redeeming their tokens before this function is called, the `usdcBalance` of the tranche will decrease. At the same time, the variable `redeemableUsdcAmount` will remain unchanged.

This behavior leads to an underflow in the process of calculating the excess:

```
require(tranche.state == TrancheState.Redeemable, "Cannot withdraw from this tranche");
uint256 excessUsdc = tranche.usdcBalance - tranche.redeemableUsdcAmount;
```

Suppose `usdcBalance` is `150` USDC, and `redeemableUsdcAmount` is `110` USDC; the admin can withdraw `excessUsdc` equal to `40` USDC. However, if users redeem their tokens first by calling the `redeem` function, the balance will drop to `usdcBalance` equal to `40` USDC. In this case, the difference `tranche.usdcBalance - tranche.redeemableUsdcAmount` will cause a revert, and the admin will be unable to withdraw the excess from the contract.

Recommendation

We recommend rethinking the process of transitioning the tranche state to `Redeemable` to ensure the admin can withdraw excess tokens without the risk of them getting stuck in the protocol.

Update

Fixed in commit 5bc426b32a85598a8e46be79cc4c9e72e6049b6d

2.2 MAJOR

M-01	Insufficient state transition checks for tranches in `Mansa`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `transitionTrancheState`	118

Description

In the function `transitionTrancheState` of the `Mansa` contract, the admin can change the tranche state to any other state (except from the `Redeemable` state) at any stage. For example, transitioning from the `Closed` state to `Redeemable` does not make sense because there would be no tokens to redeem in the tranche.

Additionally, there are no checks when transitioning a tranche from any state to `Redeemable`, even though this transition occurs once and is irreversible.

For instance, the tranche balance `tranche.usdcBalance` might be zero, or if no funds were added to the tranche during its operation, `tranche.token.totalSupply()` would also be zero. In these cases, transitioning the tranche to the `Redeemable` state is premature or meaningless.

This leads to the possibility of transitioning the tranche state from `Withdrawable` to `Redeemable` before tokens are deposited into the contract via the `repay` function. In this scenario, users would not be able to redeem their investments because the tranche state cannot be changed back from `Redeemable`, and the `repay` function only works in the `Withdrawable` state.

Recommendation

We recommend adding checks before transitioning the tranche state to `Redeemable` to avoid situations where users lose the ability to redeem their funds. Additionally, consider implementing a stricter pipeline for transitioning the tranche from one state to another.

Update

Fixed in commit 5bc426b32a85598a8e46be79cc4c9e72e6049b6d

Mansa's response:

This is not a vulnerability but rather a suggestion to prevent a careless admin that does not check the contract state before calling the state transition. Anyhow the suggestion is adopted in `5bc426b32a85598a8e46be79cc4c9e72e6049b6d`

M-02	Missing validation for `custodianAddress_` in `Mansa`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `createTranche`	61

Description

In the function `createTranche` of the `Mansa` contract, the `custodianAddress_` for a new tranche is set without any validation. Additionally, the custodian address in the created tranche cannot be changed.

This can result in an invalid custodian address being provided. For example, a null address might be passed.

Subsequently, after the tranche state is transitioned to `Withdrawable`, user funds will be transferred to the incorrect address and lost.

Recommendation

We recommend adding validation for the custodian address to avoid loss of funds.

Update

Fixed in commit 5bc426b32a85598a8e46be79cc4c9e72e6049b6d

Mansa's response:

This is not a vulnerability but rather a suggestion to prevent a careless admin from typing in `0` as custodian address in tranche creation. Anyhow the suggestion is adopted in `5bc426b32a85598a8e46be79cc4c9e72e6049b6d`

2.3 WARNING

W-01	Missing validation for `redeemRatioBip_` in `Mansa`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `createTranche`	63

Description

In the function `createTranche` of the `Mansa` contract, `redeemRatioBip_` is set for a new tranche without any validation. Additionally, the ratio in the created tranche cannot be changed.

This can result in an invalid ratio being provided. For example, a ratio of less than `10000` (less than 100%) means that users will "gift" part of their invested funds to the protocol instead of receiving income:

```
uint256 maxRedeemableUsdcAmount = tranche.redeemableTokenSupply * tranche.redeemRatioBip /
10000;
```

Recommendation

We recommend adding validation for the `redeemRatioBip_` value when creating a tranche to eliminate the possibility of setting undesirable values.

Update

Mansa's response:

This is not a vulnerability but rather a suggestion to prevent a careless admin from "fat-fingering" in a wrong ratio. The "description" of a ratio less than `10000` being "invalid" is not always in theory correct: Japan has had negative interest rate for 17 years. So this suggestion is not actually valid.

Oxorio's response:

Considering the extreme case where the interest rate is zero, users would not receive their invested funds back at all. In our view, this significantly undermines the purpose of the protocol. If such a scenario is indeed valid for the protocol, we agree with the NO ISSUE status but suggest explicitly mentioning this behavior in future documentation to avoid misleading users.

Mansa's response:

Since the ratio is not mutable after the tranche creation, user will be able to see an undesirable yield from the GUI or even in smart contract level. Thus the worst case for this issue is just bad optics and the admin wasting gas to delete the undesired tranche.

LP tokens being useful for purposes other than just receiving yield is quite common in DeFi, even the "extreme" case of `0` can theoretically be justified if the LP token can be used to get other perks in other protocol or participate in governance or airdrop.

W-02	"Dust" on balance after redemption blocks tranche deletion in `Mansa`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `redeem`	141

Description

In the function `redeem` of the `Mansa` contract, rounding occurs when dividing the value of `usdcAmount`. Therefore, the tranche's `usdcBalance` decreases by a smaller amount than required. At the same time, the number of `MansaTrancheToken` tokens for redemption `_tokenAmount` is fully burned:

```
uint256 usdcAmount = tranche.redeemableUsdcAmount * _tokenAmount /
tranche.redeemableTokenSupply;
tranche.usdcBalance -= usdcAmount;
tranche.token.adminBurnFrom(_msgSender(), _tokenAmount);
usdc.transfer(_msgSender(), usdcAmount);
```

This leads to a situation where, after burning all `MansaTrancheToken` tokens for redemption, a small amount of USDC remains in the tranche's balance, preventing the tranche from being deleted as a revert will occur when the function `deleteTranche` is called:

```
require(tranche.usdcBalance == 0, "Cannot delete tranche with non-zero USDC balance");
```

Recommendation

We recommend reconsidering the tranche deletion logic so that the remaining USDC balance in the tranche does not prevent its deletion, or that the deletion process does not rely on the tranche balance. Additionally, we recommend considering the possibility of deducting the remaining USDC from the tranche balance upon deletion.

Update

Fixed in commit 5bc426b32a85598a8e46be79cc4c9e72e6049b6d

Mansa's response:

This is a good suggestion and we have adopted it in `5bc426b32a85598a8e46be79cc4c9e72e6049b6d` to guard against people who waste their LP tokens so that we cannot remove a tranche from the UI. It does not seem like a very meaningful "attack" but nevertheless a valid annoyance that can be brought about by an attacker. Even though a simple UI change would defeat this, it would still bring about some annoyance to the team.

W-03	Possible redemption of zero USDC in `Mansa`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `redeem`	141

Description

In the function `redeem` of the `Mansa` contract, for small values of `_tokenAmount`, the result of calculating `usdcAmount` will be `0`:

```
uint256 usdcAmount = tranche.redeemableUsdcAmount * _tokenAmount /
tranche.redeemableTokenSupply;
```

For example, with the following values, we get a `usdcAmount` of zero:

- `redeemableUsdcAmount = 110 * 10^6`
- `redeemableTokenSupply = 100 * 10^18`
- `_tokenAmount = 1 * 10^11`

This leads to a meaningless redemption of `0` USDC tokens for a small `_tokenAmount`.

It also allows for spamming the frontend with events about token transfers and burns, which are emitted when the `redeem` function is called. In such a case, with a redemption `_tokenAmount` of `0`, the user will only pay for the gas.

Recommendation

We recommend considering refactoring the redemption logic to avoid zero token transfers.

Update

Fixed in commit 4005d0db4024befd8ad9994ae05f4721367f92ae

Mansa's response:

This is a valid suggestion to prevent people from burning gas to create meaningless transaction rows in the UI. Even though it would be a simple change from our UI to filter out all those, it is an annoyance nonetheless. We have adopted this suggestion in `4005d0db4024befd8ad9994ae05f4721367f92ae`

W-04	Funds of a removed user from the whitelist are locked in the contract in `Whitelist`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Whitelist.sol	contract `Whitelist` > function `remove`	18

Description

In the function `remove` of the `Whitelist` contract, the admin can remove a user from the whitelist, thereby prohibiting them from transferring tokens, investing, and redeeming in the protocol. However, the admin does not have access to the user's tokens.

If the user has already invested in a tranche, removing them from the whitelist results in freezing their tokens. The admin also cannot access these tokens. Therefore, if the user turns out to be malicious, their invested funds will be stuck in the protocol.

Recommendation

We recommend considering ways to handle the funds of users who have invested in the protocol but were removed from the whitelist.

Update

Fixed in commit 5bc426b32a85598a8e46be79cc4c9e72e6049b6d

Mansa's response:

This is a valid suggestion to cover the scenario where if we never want to re-whitelist the address forever. We have changed `deleteTranche` such that the residual funds will be transferred to the admin as a safety measure in `5bc426b32a85598a8e46be79cc4c9e72e6049b6d` to get back the USDC of the frozen users.

I-01	Absence of whitelist allows injection and distribution of "dirty" cryptocurrency in `Mansa`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `repay`	130

Description

In the function `repay` of the `Mansa` contract, any user can deposit funds into the contract, which will subsequently be distributed among users.

This function enables a malicious user to inject "dirty" cryptocurrency into the protocol and distribute it among both their own and other participants. Consequently, they "clean" a portion of funds obtained through dubious means by blending in with the crowd of protocol users.

This situation can lead to the protocol itself being banned if suspicions arise of its use for money laundering.

Recommendation

We recommend considering the possibility of restricting the set of users who have the ability to call the `repay` function.

Update

Fixed in commit 5bc426b32a85598a8e46be79cc4c9e72e6049b6d

Mansa's response:

Even though whitelisting the LP Tokens is already much stricter than rest of the DeFi ecosystems in terms of preventing "dirty" coins,

`5bc426b32a85598a8e46be79cc4c9e72e6049b6d` has incorporated this suggestion to err on the side of over paranoia.

I-02	Calculation of the difference in decimals can be moved to a constant in `Mansa`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `invest`	101

Description

In the `invest` function of the `Mansa` contract, the calculation of the `MansaTrancheToken` token amount with `decimal=18` is based on the number of USDC tokens with `decimal=6`. However, external calls are made to obtain the `decimals` values, and the calculation of their difference occurs each time the `invest` function is called:

```
uint256 tokenAmount = usdcAmount_ * 10 ** (tranche.token.decimals() - usdc.decimals());
```

Recommendation

We recommend calculating the multiplier `10 ** (tranche.token.decimals() - usdc.decimals())` once in advance and using it as a constant in the function for gas optimization and codebase cleanliness.

Update

Fixed in commit 5bc426b32a85598a8e46be79cc4c9e72e6049b6d

Mansa's response:

This is a small but valid suggestion as it does save about $0.15 in gas per transaction if on mainnet as of writing. We have adopted this suggestion in `5bc426b32a85598a8e46be79cc4c9e72e6049b6d`

I-03	Magic number in `Mansa`
Severity	**INFO**
Status	• NO ISSUE

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `transitionTrancheState`	111

Description

In the function `transitionTrancheState` of contract `Mansa` literal value with unexplained meaning are used to perform calculations.

Recommendation

We recommend defining a constant for every magic number, giving it a clear and self-explanatory name.

Update

Fixed in commit 5bc426b32a85598a8e46be79cc4c9e72e6049b6d

Mansa's response:

Supposedly this refers to `tranche.redeemRatioBip / 10000`. The fact that a "bip" is 1 / 10000 should be a common sense that it's not considered "magic number", and this number is really only used once throughout the whole contract, making the point more moot. It's like calling "100" for a percentage value a "magic number".

I-04	State `Invalid` is not used in `Mansa.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Mansa.sol	-	23

Description

In `Mansa.sol#L23`, the state `Invalid` for the tranche is not used in the protocol. The default state used is `Closed`.

Recommendation

We recommend considering removing the `Invalid` state from the set of possible tranche states, or adding logic for its use.

Update

Fixed in commit 5bc426b32a85598a8e46be79cc4c9e72e6049b6d

Mansa's response:

`Invalid` is a proxy for `0` in the enum. This is a valid suggestion and we have adopted it in `5bc426b32a85598a8e46be79cc4c9e72e6049b6d` to actually utilize it.

I-05	Floating `pragma`
Severity	**INFO**
Status	• FIXED

Description

All contracts across the codebase use the following pragma statement:

```
pragma solidity ^0.8.0;
```

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too new which has not been extensively tested.

Recommendation

We recommend locking the `pragma` to a specific version of the compiler.

Update

Fixed in commit 35f6a8c874a97ac0f885b096c82939b5ac08aa80

Mansa's response:

This is a fair suggestion and we have adopted it in `35f6a8c874a97ac0f885b096c82939b5ac08aa80`

I-06	Usage of string instead of `uint128` for storing UUID in `Mansa`
Severity	**INFO**
Status	• NO ISSUE

Location

File	Location	Line
Mansa.sol	contract `Mansa`	33

Description

In the contract `Mansa`, a UUID in the form of a string is used for identifying a tranche. However, UUID version 4 is typically represented numerically and can fit into a `uint128`:

> An implementation may generate 128 bits of random data that is used to fill out the UUID fields

Recommendation

We recommend considering the use of the `uint128` type instead of the `string` type for the UUID identifier of the tranche to optimize gas usage. Additionally, this would eliminate the need for validating and maintaining the string format of the UUID.

Update

Mansa's response:

This is an interesting suggestion but we do not feel it is necessary at the moment. Even though the variable is named "UUID" we intend for it to be any strings to keep it flexible.

I-07	Unused `duration` field in `Mansa`
Severity	**INFO**
Status	• NO ISSUE

Location

File	Location	Line
Mansa.sol	contract `Mansa` > function `createTranche`	57

Description

In the function `createTranche` of contract `Mansa`, a `duration` field is set when creating a new tranche, but it is not used anywhere thereafter.

Recommendation

We recommend considering the removal of the `duration` field from the tranche structure or describing the logic for its utilization.

Update

Mansa's response:

The `duration` is to be read from the UI. The variable is required for investors to calculate the APY along with `redeemRatioBip`. So this suggestion is not applicable for our use case.

3 APPENDIX

OXORIO

3.1 DISCLAIMER

At the request of client, Oxorio consents to the public release of this audit report. The information contained in this audit report is provided "as is," without any representations or warranties whatsoever. Oxorio disclaims any responsibility for damages that may arise from or in relation to this audit report. Oxorio retains copyright of this report.

The audit makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about the fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

3.2 SECURITY ASSESSMENT METHODOLOGY

Oxorio's smart contract audit methodology is designed to ensure the security, reliability, and compliance of smart contracts throughout their development lifecycle. Our process integrates the Smart Contract Security Verification Standard (SCSVS) with our advanced techniques to address complex security challenges. For a detailed look at our approach, please refer to the full version of our methodology. Here is a concise overview of our auditing process:

1. Project Architecture Review

All necessary information about the smart contract is gathered, including its intended functionality and dependencies. This stage sets the foundation by reviewing documentation, business logic, and initial code analysis.

2. Vulnerability Assessment

This phase involves a deep dive into the smart contract's code to identify security vulnerabilities. Rigorous testing and review processes are applied to ensure robustness against potential attacks.

This stage is focused on identifying specific vulnerabilities within the smart contract code. It involves scanning and testing the code for known security weaknesses and patterns that could potentially be exploited by malicious actors.

3. Security Model Evaluation

The smart contract’s architecture is assessed to ensure it aligns with security best practices and does not introduce potential vulnerabilities. This includes reviewing how the contract integrates with external systems, its compliance with security best practices, and whether the overall design supports a secure operational environment.

This phase involves a analysis of the project's documentation, the consistency of business logic as documented versus implemented in the code, and any assumptions made during the design and development phases. It assesses if the contract's architectural design adequately addresses potential threats and integrates necessary security controls.

4. Cross-Verification by Multiple Auditors

Typically, the project is assessed by multiple auditors to ensure a diverse range of insights and thorough coverage. Findings from individual auditors are cross-checked to verify accuracy and completeness.

5. Report Consolidation

Findings from all auditors are consolidated into a single, comprehensive audit report. This report outlines potential vulnerabilities, areas for improvement, and an overall assessment of the smart contract's security posture.

6. Reaudit of Revised Submissions

Post-review modifications made by the client are reassessed to ensure that all previously identified issues have been adequately addressed. This stage helps validate the effectiveness of the fixes applied.

7. Final Audit Report Publication

The final version of the audit report is delivered to the client and published on Oxorio's official website. This report includes detailed findings, recommendations for improvement, and an executive summary of the smart contract's security status.

3.3 CODEBASE QUALITY ASSESSMENT REFERENCE

The tables below describe the codebase quality assessment categories and rating criteria used in this report.

Category	Description
Access Control	Evaluates the effectiveness of mechanisms controlling access to ensure only authorized entities can execute specific actions, critical for maintaining system integrity and preventing unauthorized use.
Arithmetic	Focuses on the correct implementation of arithmetic operations to prevent vulnerabilities like overflows and underflows, ensuring that mathematical operations are both logically and semantically accurate.
Complexity	Assesses code organization and function clarity to confirm that functions and modules are organized for ease of understanding and maintenance, thereby reducing unnecessary complexity and enhancing readability.
Data Validation	Assesses the robustness of input validation to prevent common vulnerabilities like overflow, invalid addresses, and other malicious input exploits.
Decentralization	Reviews the implementation of decentralized governance structures to mitigate insider threats and ensure effective risk management during contract upgrades.
Documentation	Reviews the comprehensiveness and clarity of code documentation to ensure that it provides adequate guidance for understanding, maintaining, and securely operating the codebase.
External Dependencies	Evaluates the extent to which the codebase depends on external protocols, oracles, or services. It identifies risks posed by these dependencies, such as compromised data integrity, cascading failures, or reliance on centralized entities. The assessment checks if these external integrations have appropriate fallback mechanisms or redundancy to mitigate risks and protect the protocol's functionality.
Error Handling	Reviews the methods used to handle exceptions and errors, ensuring that failures are managed gracefully and securely.
Logging and Monitoring	Evaluates the use of event auditing and logging to ensure effective tracking of critical system interactions and detect potential anomalies.
Low-Level Calls	Reviews the use of low-level constructs like inline assembly, raw `call` or `delegatecall`, ensuring they are justified, carefully implemented, and do not compromise contract security.

Category	Description
Testing and Verification	Reviews the implementation of unit tests and integration tests to verify that codebase has comprehensive test coverage and reliable mechanisms to catch potential issues.

3.3.1 Rating Criteria

Rating	Description
Excellent	The system is flawless and surpasses standard industry best practices.
Good	Only minor issues were detected; overall, the system adheres to established best practices.
Fair	Issues were identified that could potentially compromise system integrity.
Poor	Numerous issues were identified that compromise system integrity.
Absent	A critical component is absent, severely compromising system safety.
Not Applicable	This category does not apply to the current evaluation.

3.4 FINDINGS CLASSIFICATION REFERENCE

3.4.1 Severity Level Reference

The following severity levels were assigned to the issues described in the report:

Title	Description
CRITICAL	Issues that pose immediate and significant risks, potentially leading to asset theft, inaccessible funds, unauthorized transactions, or other substantial financial losses. These vulnerabilities represent serious flaws that could be exploited to compromise or control the entire contract. They require immediate attention and remediation to secure the system and prevent further exploitation.
MAJOR	Issues that could cause a significant failure in the contract's functionality, potentially necessitating manual intervention to modify or replace the contract. These vulnerabilities may result in data corruption, malfunctioning logic, or prolonged downtime, requiring substantial operational changes to restore normal performance. While these issues do not immediately lead to financial losses, they compromise the reliability and security of the contract, demanding prioritized attention and remediation.
WARNING	Issues that might disrupt the contract's intended logic, affecting its correct functioning or making it vulnerable to Denial of Service (DDoS) attacks. These problems may result in the unintended triggering of conditions, edge cases, or interactions that could degrade the user experience or impede specific operations. While they do not pose immediate critical risks, they could impact contract reliability and require attention to prevent future vulnerabilities or disruptions.
INFO	Issues that do not impact the security of the project but are reported to the client's team for improvement. They include recommendations related to code quality, gas optimization, and other minor adjustments that could enhance the project's overall performance and maintainability.

3.4.2 Status Level Reference

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

Title	Description
NEW	Waiting for the project team's feedback.

Title	Description
FIXED	Recommended fixes have been applied to the project code and the identified issue no longer affects the project's security.
ACKNOWLEDGED	The project team is aware of this finding. Recommended fixes for this finding are planned to be made. This finding does not affect the overall security of the project.
NO ISSUE	Finding does not affect the overall security of the project and does not violate the logic of its work.

3.5 ABOUT OXORIO

OXORIO is a blockchain security firm that specializes in smart contracts, zk-SNARK solutions, and security consulting. With a decade of blockchain development and five years in smart contract auditing, our expert team delivers premier security services for projects at any stage of maturity and development.

Since 2021, we've conducted key security audits for notable DeFi projects like Lido, 1Inch, Rarible, and deBridge, prioritizing excellence and long-term client relationships. Our co-founders, recognized by the Ethereum and Web3 Foundations, lead our continuous research to address new threats in the blockchain industry. Committed to the industry's trust and advancement, we contribute significantly to security standards and practices through our research and education work.

Our contacts:

- oxor.io
- ping@oxor.io
- Github
- Linkedin
- Twitter

THANK YOU FOR CHOOSING