yAudit Temple DAO Origami Recheck Review

Review Resources:

- Some internal docs and visuals were provided

Auditors:

- pandadefi
- spalen

Table of Contents

Review Summary

Temple DAO Origami

The goal of Temple DAO's Origami product is to offer auto-compounded yield offerings on underlying strategies, maximising returns without sacrificing liquidity. The first strategy being offered is on GLP and GMX. Origami will be deployed on Arbitrum and Avalanche, the two chains where GMX is deployed, even though the current Temple core contracts are deployed on Ethereum.

The contracts of the Temple DAO Origami Repo were reviewed over 5 days. The code review was performed by 2 auditors between February 21 and February 26, 2023. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit 5ddb424ca1cba8698a51fc21510c1891598a7f09 for the Temple DAO repo.

This was the second review of the Temple DAO Origami Repo.

Temple DAO fixed the issues from the first report so yAudit did a review of these mitigations by comparing changes made from commit a31d192ab54ca7d21f2dee30c630a6ec1843b646, which was the commit of the first review, to commit 5ddb424ca1cba8698a51fc21510c1891598a7f09, which was the commit that included the mitigations.

Scope

The scope of the review consisted of the files in the following directories at the specific commit:

- contracts/investments/*
- contracts/common/*

After the findings were presented to the Temple DAO team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Temple DAO and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Average	Main functions are limited to Governable contract. Protocol flow is guarded using Operators. Additionally, protocol pausing is enabled to pausers.

Category	Mark	Description
Mathematics	Good	No complex math is performed in the contracts. Only basic accounting and value management is done.
Complexity	Average	The complexity of the Temple Origami contracts were on par with a typical DeFi project. Some of the key functionalities included integration with a yield source (GMX), compounding of rewards, and custom tokens to track value.
Libraries	Good	The only external dependencies are OpenZeppelin libraries.
Decentralization	Average	The Governable role has significant privileges in the Temple Origami contracts. The Operator and _minters are also privileged roles that may reduce the decentralization of the system.
Code stability	Good	The latest changes are addressing only issues from the prior report.
Documentation	Good	Visuals were created to show the main user flows. Clear NatSpec comments existed throughout the contracts.
Monitoring	Good	Events were emitted by all key functions that perform state variable changes.
Testing and verification	Good	All tests are passing with 100% code coverage.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements

- Gas savings
    - Findings that can improve the gas efficiency of the contracts
- Informational
    - Findings including recommendations and best practices

Critical Findings

None.

High Findings

1. High - Incorrect transfer of funds on token exit

When exiting the vault token, the user can define the recipient address to get the funds, but this address is ignored, and all funds are sent to the message sender instead.

Technical Details

Calling `exitToToken(ExitQuoteData calldata quoteData, address recipient)` user can define address recipient to receive the funds but this address is ignored and all funds are sent to the message sender instead.

Impact

High. This can lead to loss of funds.

Recommendation

Use defined recipient address instead of `msg.sender` at L325.

```
- toTokenAmount = _redeemReservesFromShares(
-          quoteData.investmentTokenAmount,
-          msg.sender,
-          quoteData.minToTokenAmount,
-          msg.sender
- );
+ toTokenAmount = _redeemReservesFromShares(
+          quoteData.investmentTokenAmount,
+           msg.sender,
+           quoteData.minToTokenAmount,
+           recipient
+ );
```

Add the test to cover the case when the recipient is different from message sender: https://gist.github.com/spalen0/1bfc6379f00a9d73a1285367e8ccd669.

Developer Response

@frontier159: Fixed in commit ab2a428

Medium Findings

None.

Low Findings

None.

Gas Savings Findings

1. Gas - Avoid double checking

Removing double-checking of the same condition will save gas.

Technical Details

At L222 there is a check for condition `reserveTokenAmount != 0`, which already checked at L213.

Impact

Gas savings.

Recommendation

Change condition at L222:

```
- if (receiver != address(this) && reserveTokenAmount != 0) {
+ if (receiver != address(this)) {
```

Developer Response

@frontier159: Fixed in commit d26ca15

Informational Findings

1. Informational - Incorrect comment

The comment about function accessibility is incorrect.

Technical Details

File MintableToken has a comment with an incorrect function accessibility.

Impact

Informational.

Recommendation

Update comment with governance accessibility.

Developer Response

@frontier159: Fixed in commit 3ab71c8

2. Informational - Governable and GovernableUpgradeable should abstract common code into a common contract

The two contracts are sharing the same code. `Governable.sol` `GovernableUpgradeable.sol` can inherit a GovernableBase contract.

Impact

Informational.

Technical Details

By reducing code duplication, the codebase will be easier to maintain and less error-prone. Both contracts can inherit from a common contract that contains the common code.

https://github.com/TempleDAO/origami/blob/5ddb424ca1cba8698a51fc21510c1891598a7f09/apps/protocol/contracts/common/access/Governable.sol https://github.com/TempleDAO/origami/blob/5ddb424ca1cba8698a51fc21510c1891598a7f09/apps/protocol/contracts/common/access/GovernableUpgradeable.sol

Recommendation

Add a GovernableBase contract that contains the common code and make Governable and GovernableUpgradeable inherited from it.

Developer Response

@frontier159: Fixed in commit ebeedc1

Final remarks

All the changes correctly address the issues in the prior report. Adding operators is limited to governance which is a good base for protocol decentralization. This is important because, with the current flow, operators have the ability to extract value from the protocol, as stated in the previous report. The code base is well prepared with all tests passing and maximum code coverage.