# yAudit Temple DAO Origami Review

**Review Resources:**

- Some internal docs and visuals were provided

**Auditors:**

- pandadefi
- spalen
- engn33r

## 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 21 days. The code review was performed by 3 auditors between January 23, 2023 and February 12, 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 a31d192ab54ca7d21f2dee30c630a6ec1843b646 for the Temple DAO repo.

## 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 | Some privileged roles exist in the contracts, such as the Operator role and the CAN_MINT role. Many functions are protected by the onlyOwner modifier. |
| Mathematics | Good | No complex math is performed in the contracts. Only basic accounting and value management is done. |

| Category | Mark | Description |
| --- | --- | --- |
| 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 owner role has significant privileges in the Temple Origami contracts. The Operator and CAN_MINT roles are also privileged roles that may reduce the decentralization of the system. The `recoverToken()` function in many contracts indicates a high level of trust in the contract owner. |
| Code stability | Average | The code was nearly production ready but may not have been completely frozen prior to production deployment. |
| 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 | Low | The tests did not fully execute without errors so the extent of the test coverage could not be checked. |

## 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 - Harvesting vault can be front-run for profit

`sharesToReserves` increases immediately on profit distribution. A bad actor can sandwich the `harvestRewards()` call for immediate profit.

#### Technical Details

Calling `harvestRewards()` will collect and distribute rewards. A bad actor can deposit tokens into the ovToken before the harvest and withdraw from ovToken right after. The `sharesToReserves` value will instantaneously increase and the attacker will be able to withdraw more tokens than deposited with reduced incentives for users to invest in the protocol in the future.

#### Impact

High. The protocol harvest call can be front-run, reducing the benefits for people invested in the protocol.

#### Recommendation

Distribute rewards over hours instead of immediate sharesToReserves increase. One option is to keep a queue of rewards to distribute overtime. You can find an example in the yearn vault.

#### Developer Response

@frontier159: Fixed in commit 8cbf5fb

The current vaults have a higher cost to enter/exit than what one would get by front-running. However point taken, and a good thing to address.

The plan to remediate is to add 'per second' distribution to users within the base `RepricingToken` contract. Summary (some detail left out for brevity):

1. When `addReserves()` is called, pull the reserve tokens from the caller
2. Those reserve tokens aren't immediately all available to users - it's dripped in per second.
3. So `totalReserves` then becomes a function instead of just a counter, effectively `return actualisedReserves + accruedReserves`
4. Whenever new reserves are added, we actualise:
    a. Any accrued up to now is added to `actualisedReserves`
    b. Any left over balance from the amount previously added is added to the new reserves being added
    c. The distribution timer restarts (set to block.timestamp)

We are planning on dripping the rewards in over a period of a week, and harvest daily. So each day the timer on the distribution will restart but the left over is carried over and the timer restarted.

## Medium Findings

### 1. Medium - `_handleGmxRewards()` returned values can lead to wrong accounting

When claiming GMX rewards, rewards come from GMX and GLP pools. The accounting can be wrong if the tokens claimed are staked.

#### Technical Details

The function `_handleGmxRewards()` calculates GMX rewards based on balance changes and claimable rewards from GLP without considering if the claimed tokens are staked.

When staking rewards, the code doesn't set to zero `esGmxFromGlp`. This is inconsistent with `esGmxFromGmx` computed using a balance change `esGmxFromGlp` and will be zero if rewards get staked.

OrigamiGmxEarnAccount.sol#L396-L424

#### Impact

Medium. Inaccurate accounting may happen under certain circumstances.

#### Recommendation

Make sure to set `esGmxFromGlp` to zero when `shouldStakeEsGmx` is `true`.

#### Developer Response

@frontier159: Fixed in commit fe96c8a

## Low Findings

### 1. Low - Use `glpRewardRouter` for fetching glp trackers

Initializer should use `glpRewardRouter` for fetching GLP trackers but uses `gmxRewardRouter`.

#### Technical Details

Currently, both routers point to the same trackers, but this could change. Deployed `glpRewardRouter`, for GMX trackers aren't set, points to address 0. The same could happen for `gmxRewardRouter`, GLP trackers could point to address 0.

#### Impact

Low. Trackers could be set to address 0 and break some contract functionalities.

#### Recommendation

Use values from `glpRewardRouter` for setting variables `stakedGlpTracker` and `feeGlpTracker`.

```
stakedGlpTracker = IGmxRewardTracker(glpRewardRouter.stakedGlpTracker());
feeGlpTracker = IGmxRewardTracker(glpRewardRouter.feeGlpTracker());
```

#### Developer Response

@frontier159: Fixed in commit 5925173

### 2. Low - No Chainlink staleness check in `oraclePrice()`

`oraclePrice()` retrieves price data from Chainlink, but there are no checks to discard data if the oracle returns stale data.

#### Technical Details

The Chainlink `latestRoundData()` function returns price data along with the roundId and timestamp of the data. If the data is stale, it should be discarded. Otherwise the protocol

will trust outdated data that could lead to a loss of value from using an inaccurate exchange rate. It is recommended to check the roundId and timestamp values that the oracle returns, as shown in other security report findings here and here.

**Impact**

Low. The Chainlink oracle data should be checked for staleness.

**Recommendation**

Consider modifying `oraclePrice()` to the following:

```
          function oraclePrice(address _oracle) public view returns (uint256 price) {
        IAggregatorV3Interface oracle = IAggregatorV3Interface(_oracle);
-        (, int256 feedValue, , , ) = oracle.latestRoundData();
+        (uint80 roundId, int256 feedValue, , uint256 updatedAt, uint80
answeredInRound) = oracle.latestRoundData();
+                    if (answeredInRound <= roundId && block.timestamp - updatedAt >
ORACLE_STALENESS_THRESHOLD) revert InvalidPrice(feedValue);
        if (feedValue < 0) revert InvalidPrice(feedValue);
        price = scaleToPrecision(uint256(feedValue), oracle.decimals());
    }
```

**Developer Response**

@frontier159: Fixed in commit 011c36d

## Gas Savings Findings

### 1. Gas - Variables could be immutables

Some variables set on `OrigamiGmxEarnAccount` aren't likely to change. These variables can be set during contract creation and the variables can be declared immutable for gas savings.

**Technical Details**

These variables can be declared immutable:

```
/// @notice $GMX
```

```
IERC20Upgradeable public gmxToken;

/// @notice $esGMX - escrowed GMX
IERC20Upgradeable public esGmxToken;

/// @notice $wrappedNative - wrapped ETH/AVAX
IERC20Upgradeable public wrappedNativeToken;
```

These three variables won't change. The contract constructor can set them appropriately.

#### Impact

Gas savings.

#### Recommendation

Declare these three variables as immutable. The contract constructor can take `gmxRewardRouter` to fetch the values.

#### Developer Response

@frontier159: Fixed in commit 802f3c4

### 2. Gas - Initialize variable only if needed

In some cases where variables are initialized, the variables won't be used. This in inefficient and variables should only be initialized when they are used.

#### Technical Details

Variable `esGmxReinvested` is initialized before the if statement but it's only used inside the if block.

#### Impact

Gas savings.

#### Recommendation

Initialize variable [`esGmxReinvested`] inside the if block.

```
-  uint256 esGmxReinvested;
  if (totalEsGmxClaimed != 0) {
+      uint256 esGmxReinvested;
```

#### Developer Response

@frontier159: Fixed in commit 2e575b3

### 3. Gas - Reuse local variable

Local variables can be reused instead of initializing new ones without losing code readability.

#### Technical Details

Variable `fromToken` can be reused instead of initializing the new variable `tokenIn`. The same applies to variable `tokenOut`.

Local variable `reserveAmount` can be dropped from here and here if inline is used like this: `underlyingQuoteData.underlyingExitQuoteData.investmentTokenAmount = _redeemReservesFromShares`

#### Impact

Gas savings.

#### Recommendation

Reuse existing variables:

```
- address tokenOut = (toToken == address(0)) ? wrappedNativeToken : toToken;
+ toToken = (toToken == address(0)) ? wrappedNativeToken : toToken;
```

#### Developer Response

@frontier159: Fixed in commit 8e2bd54

### 4. Gas - Use `msg.sender` not `owner()`

When two variables or function calls return equivalent values, it makes sense to use the option that uses less gas.

#### Technical Details

It is cheaper to call `msg.sender` instead of `ownable()` when they both return the same value. If this change is made in the constructor of MintableToken, the range of gas used on the deployment of MintableToken is reduced from the original range of 2296764-2296884 to 2296354-2296474, saving roughly 400 gas.

#### Impact

Gas savings.

#### Recommendation

Replace `owner()` with `msg.sender` in the MintableToken constructor.

#### Developer Response

@frontier159: Fixed in commit b538b51

## Informational Findings

### 1. Informational - Incorrect comment

The comment has an incorrect file name.

#### Technical Details

File OrigamiGmxInvestment has a comment with an incorrect file name.

#### Impact

Informational.

#### Recommendation

Update comment to correspond to the file name.

#### Developer Response

@frontier159: Fixed in commit 7b78684

### 2. Informational - Oracles price can be exploited

Two price oracles that are used for off-chain calculations can be manipulated. These oracles should never be used for on-chain calculations.

#### Technical Details

- TokenPrices.sol#L74: this price oracle can be exploited with a single block sandwich attack.
- TokenPrices.sol#L85: this price oracle can be exploited via a multi-block attack by block producers. More info.

#### Impact

Informational.

#### Recommendation

Make sure not to use these two oracles from a smart contract.

#### Developer Response

@frontier159: Fixed in commit fb002bf

### 3. Informational - Update comment to NatSpec format

Some comments are written as NatSpec but are missing characters, including `/` and `@notice`, to be in the correct format.

#### Technical Details

In file OrigamiGmxManager variables `primaryEarnAccount` and `secondaryEarnAccount` could be in NatSpec format.

At least two comments (1, 2) are missing the @notice NatSpec tag.

Finally, the comment on `reservesToShares()` is identical to the comment on `sharesToReserves()`, which is incorrect. The comment for `reservesToShares()` should be reversed to read “How many shares given a number of reserve tokens”.

#### Impact

Informational.

#### Recommendation

Update comments to be in NatSpec format by adding missing characters.

#### Developer Response

@frontier159: Fixed in commit 8be8c94

### 4. Informational - Verify fees and rewards addresses

The addresses for receiving fees and rewards can be set to address 0, so all fees and rewards could be lost.

#### Technical Details

Setter functions in OrigamiGmxManager for `feeCollector` and rewards aggregators doesn’t verify input for the default 0 value. There are no checks to prevent an address of 0 when the fees and rewards are distributed.

#### Impact

Informational. Only the owner can set the addresses, so it is under the owner's control, but could lead to lost funds for protocol users.

#### Recommendation

Verify the addresses are not 0 before setting state variables and setting default values in the constructor, could use address(this). Another option is to keep fees and rewards in the OrigamiGmxManager if the address is not set and recover the token later.

#### Developer Response

@frontier159: Fixed in commit 70dd257

Won't fix checking in the constructor - we have robust checks on mainnet deploys, risk here is accepted.

### 5. Informational - Remove `removeReserves(uint256 amount)`

The function `removeReserves(uint256 amount)` exposes a possibility to drain the protocol, but the function doesn't have a use case.

#### Technical Details

The function enables operators to take all `reserveToken` which can after be redeemed for other tokens depending on the OrigamiInvestment implementation. Even `recoverToken(address _token, address _to, uint256 _amount)` function, which is limited to only the owner, verifies the owner cannot drain the protocol.

#### Impact

Informational.

#### Recommendation

Remove the function.

#### Developer Response

@frontier159: Fixed in commit 6ea7a6c

### 6. Informational - Trader Joe AMM is moving liquidity to a new AMM design

The Trader Joe AMM is moving liquidity to a new AMM design so it would benefit TokenPrices to use the newer AMM.

#### Technical Details

Trader Joe is used as a price oracle on TokenPrices.sol#L74. Trader Joe announced a new AMM design with breaking ABI changes. The design will allow anyone, not just Trader Joe, to create new trading pools, so liquidity is expected to move to the new AMM.

- announcement
- doc

#### Impact

Informational.

#### Recommendation

Use the newer Trader Joe AMM. Replace `joePair.getReserves()` with `joePair.getReservesAndId()`.

#### Developer Response

@frontier159: Fixed in commit d7499c8

Now using the v2 helper to get the 'best' quote from all v1 and v2 pools.

### 7. Informational - Incorrect NatSpec

The Operators.sol contract has a NatSpec error.

#### Technical Details

On Operators.sol#L19 `@dev` NatSpec specifies this `__Operators_init()` initializes the owner, but it's not initializing the owner.

#### Impact

Informational.

#### Recommendation

Update NatSpec in Operators.sol.

#### Developer Response

@frontier159: Fixed in commit 65b27c6

### 8. Informational - `addToReserveAmount` could be a percentage value

The `addToReserveAmount` uint256 value in `HarvestGmxParams` and `HarvestGlpParams` structures can be expressed as a percentage value for more precision.

#### Technical Details

In `_compoundOvGmxRewards()` and `_compoundOvGlpRewards()`, the number of tokens to add to the reserve could be calculated using the returned value from `investWithToken()` and a percentage. This change would improve the precision of tokens added to the reserve, making it easier to send 100% of the rewards after slippage to the reserve.

#### Impact

Informational.

#### Recommendation

Use a percent-based approach using the return value from `investWithToken()`

#### Developer Response

@frontier159: Fixed in commit 8cbf5fb

### 9. Informational - Replace deprecated dependency

MintableToken has a dependency of draft-ERC20Permit.sol, but this dependency is described by OpenZeppelin as deprecated.

#### Technical Details

draft-ERC20Permit.sol is the old file in @openzeppelin/contracts which has been replaced with ERC20Permit.sol. Remove the import of draft-ERC20Permit.sol and instead import ERC20Permit.sol.

A related simplification is the ERC20.sol dependency can be removed from MintableToken because it is already imported through ERC20Permit.sol.

#### Impact

Informational.

#### Recommendation

Import ERC20Permit.sol instead of draft-ERC20Permit.sol in MintableToken. Remove the ERC20.sol import from MintableToken.

#### Developer Response

@frontier159: Nothing to fix

This hasn't yet been released – it will be in 4.9.*, where as the released version is 4.8.1

NB: `master` branch is their yet to be released version. See `release-v4.8` branch for the v4.8.* versions ie: (https://github.com/OpenZeppelin/openzeppelin-contracts/tree/release-v4.8/contracts/token/ERC20/extensions)

## 10. Informational - Unusual Operator.sol implementation

The Operator.sol is implemented similar to an upgradeable contract from openzeppelin-contracts-upgradeable, but it is used as a dependency in contracts that are not upgradeable. Only OrigamiGmxEarnAccount is an upgradeable contract behind a proxy, the other contracts that inherit Operator are not upgradeable.

#### Technical Details

The Operator.sol contract is implemented in the same pattern as contracts from openzeppelin-contracts-upgradeable. This includes inheriting Initializeable and having an init function. But unlike other OZ upgradeable contracts, the init functions in Operator.sol don't do anything. There is no difference in the contract if it is initialize or not.

A side effect of how this contract is used by other contracts is that every contract that inherits Operator.sol will have its own list of operators. If the intent is to manage only a single list of operators that have access to several different contracts, then consider deploying Operator.sol on its own, rather than as a dependency, and integrate it with the other contracts accordingly.

#### Impact

Informational.

#### Recommendation

Consider modifying Operator.sol to remove unnecessary artifacts borrowed from the openzeppelin-contracts-upgradeable pattern.

#### Developer Response

@frontier159: Fixed in commit 65b27c6

## 11. Informational - Reconsider using DEFAULT_ADMIN_ROLE

The DEFAULT_ADMIN_ROLE role in AccessControl is effectively a superuser role. It may make sense to avoid using this role if the goal is to make the contract more decentralized and less reliant on trusting a specific address.

#### Technical Details

OpenZeppelin's documentation for DEFAULT_ADMIN_ROLE warns that the role is effectively a superuser. If the only changing of roles is through `addMinter()` and `revokeRole()`, using DEFAULT_ADMIN_ROLE and importing AccessControl may be overkill. It could be simpler to maintain a mapping of addresses that have this access instead of inheriting the library.

Related to this, `addMinter()` and `removeMinter()` have duplicate modifiers. In the existing code, the caller must be the owner because of the modifier in MintableToken and the caller must be the adminRole because of the modifier in AccessControl. Consider removing the onlyOwner modifier to save gas.

#### Impact

Informational.

#### Recommendation

Remove the onlyOwner modifier to save gas. Consider whether inheriting AccessControl is necessary at all or whether using a local mapping of addresses that can mint is a viable replacement.

#### Developer Response

@frontier159: Fixed in commit b538b51

### 12. Informational - Consider zero for minAmount

`mintAndStakeGlp()` in OrigamiGmxEarnAccount has two minAmount arguments. One of these can be removed and a zero value passed to `glpRewardRouter.mintAndStakeGlp()`.

#### Technical Details

`glpRewardRouter.mintAndStakeGlp()` has two minAmount arguments. Only one of these is really necessary. Consider removing the other and replacing it with a zero minAmount depending on the standard use case for the `mintAndStakeGlp()` function.

#### Impact

Informational.

#### Recommendation

Remove the `minUsdg` or `minGlp` argument from `mintAndStakeGlp()` and pass a zero minAmount instead of this value.

#### Developer Response

@frontier159: Fixed in commit a2037d8

### 13. Informational - Broken link

There is a broken link in a comment.

#### Technical Details

TokenPrices.sol links to https://docs.uniswap.org/sdk/guides/fetching-prices which returns Page Not Found. Consider linking to the archived page https://web.archive.org/web/20210918154903/https://docs.uniswap.org/sdk/guides/fetching-prices.

#### Impact

Informational.

#### Recommendation

Fix the broken link.

#### Developer Response

@frontier159: Fixed in commit 8be8c94

### 14. Informational - Typo

At least one comment has a typo.

#### Technical Details

adggregator -> aggregator

#### Impact

Informational.

#### Recommendation

Fix typos.

#### Developer Response

@frontier159: Fixed in commit 4a8c036

## Final remarks

### spalen

In summary, complexity should be reduced to enhance security, and some aspects of Origami may benefit from refactoring to simplify the overall design. For example, the flow of depositing and withdrawing is split into multiple steps between different contracts. The process starts with the vault to invest funds, then value flows to the manager, and it ends up in the earn account. The flow is complex and guarded by operators, but there are potential problems in the withdrawal process if there are malicious operators. There are several withdrawal functions that could remove significant value from the protocol, found in the earn account contract function, unstake GMX, and in manager contract. Finally, the earn account contract is upgradeable and holds all assets, which puts a lot of faith in the protocol owners. This can be mitigated by providing a clear upgrade procedure with a timelock and limiting the contract operators.

### pandadefi

No critical risks were found. The code is well structured allowing it to be future-proof to follow GMX possible changes. Some of the profit distribution mechanism needs to be rethought to prevent exploits. There is a large number of functions with protected access, and operating those will require caution.

### engn33r

Integrations with GMX are usually a bit more complex that older DeFi protocols because of the different tokens and contracts involved, and this case is no different. Because GMX is dynamic and has frequent changes, like the removal of some cooldown parameters a few months ago, integrations with GMX must be designed for adaptation. The choice to use a custom vault that is not ERC4626 compliant is a bit unusual these days, but the vault appears to do its job properly so it doesn't matter much. Providing a token to compound GMX rewards while abstracting away the complexity of the protocol has the potential for use cases even outside of the TEMPLE token.