Origami Boyco Vault

Audit Report

prepared by Pavel Anokhin (@panprog)

January 22, 2025

Version: 1

Contents

1. About Guardefy and @panprog

Pavel Anokhin or **@panprog**, doing business as Guardefy, is an independent smart contract security researcher with a track record of finding numerous issues in audit contests, bugs bounties and private solo audits. His public findings and results are available at the following link:

https://audits.sherlock.xyz/watson/panprog

2. Disclaimer

Smart contract audit is a time, resource and expertise bound effort which doesn't guarantee 100% security. While every effort is put into finding as many security issues as possible, there is no guarantee that all vulnerabilities are detected nor that the code is secure from all possible attacks. Additional security audits, bugs bounty programs and on-chain monitoring are strongly advised.

This security audit report is based on the specific commit and version of the code provided. Any modifications in the code after the specified commit may introduce new issues not present in the report.

3. Scope of the audit

The code at the following link was reviewed:

https://github.com/TempleDAO/origami

commit hash: af1ee5fc0a73e9674760abf0393f76a10005dbdc

Files in scope:

```
apps/protocol/contracts/investments/bera/OrigamiBoycoVault.sol
apps/protocol/contracts/investments/bera/OrigamiBoycoUsdcManager.sol
apps/protocol/contracts/common/bera/OrigamiBeraRewardsVaultProxy.sol
apps/protocol/contracts/common/bera/OrigamiBeraBgtProxy.sol
apps/protocol/contracts/common/balancer/OrigamiBalancerComposableStablePoolHelper.sol
apps/protocol/contracts/common/access/OrigamiElevatedAccess.sol
apps/protocol/contracts/common/access/OrigamiElevatedAccessBase.sol
apps/protocol/contracts/common/access/OrigamiElevatedAccessUpgradeable.sol
apps/protocol/contracts/common/OrigamiErc4626.sol
apps/protocol/contracts/investments/util/OrigamiManagerPausable.sol
apps/protocol/contracts/libraries/OrigamiMath.sol
```

4. Audit Timeline

Audit start: January 15, 2025

Audit report delivered: January 22, 2025

Fixes reviewed: January 22, 2025

5. Findings

5.1. High severity findings

None found.

5.2. Medium severity findings

None found.

5.3. Low severity findings

5.3.1. [L-1] `OrigamiBoycoUsdcManager.setBexPoolHelper` doesn't change `bexLpToken` and `_usdcIndex` which will cause incorrect accounting when deploying liquidity.

OrigamiBoycoUsdcManager constructor sets the `bexPoolHelper` and initializes `bexLpToken` and `_usdcIndex` from `bexPoolHelper`. The issue is that admin can change `bexPoolHelper` via `setBexPoolHelper` function. However, both `bexLpToken` and `_usdcIndex` do not change in such case, keeping the old values. If the new `bexPoolHelper` has different underlying pool, the manager contract will still use the values from the previous pool, causing liquidity deployment in incorrect token / incorrect amounts or reverts.

Likelihood

Low, only when admin changes `bexPoolHelper` to a new helper with new underlying pool.

Impact

When admin deploys liquidity, incorrect token or incorrect amount of token is used or transaction reverts.

Possible mitigation

Move initialization from `bexPoolHelper` into `setBexPoolHelper` and use it in the constructor.

Status: Fixed

Fix Review: Fixed as suggested. Note: since `bexLpToken` and `_usdcIndex` are cached values, in unlikely case underlying balancer pool tokens change, these values will have to be updated (by calling `setBexPoolHelper` with the same address).

5.3.2. [L-2] `OrigamiErc4626.seedDeposit` can be called with `assets = 0` to set `maxTotalSupply` while `totalSupply` is still 0.

`OrigamiErc4626` implementation uses `seedDeposit` which can be called only by admin to prevent inflation attack. All the other deposits before the seed deposit are prevented by `maxTotalSupply`, which is initialized with 0 and thus doesn't allow any deposits.

Admin can set maxTotalSupply via `seedDeposit` or via `setMaxTotalSupply`. However, `setMaxTotalSupply` can not be called before the first deposit (when `totalSupply == 0`).

The issue is that admin can still call seedDeposit with assets deposited set to 0, setting maxTotalSupply to any value without any deposit and bypassing the setMaxTotalSupply require. The other users will then be able to deposit / execute inflation attack.

Likelihood

Low since this is admin action and admin is not supposed to deposit 0 assets with seedDeposit. However, it's still possible, for example as an admin mistake, and allows to bypass the setMaxTotalSupply require.

Impact

Inflation attack can be performed by any user.

Possible mitigation

Require seedDeposit assets to be above 0.

Status: Fixed

Fix Review: Fixed as suggested.

5.3.3. [L-3] OrigamiBalancerComposableStablePoolHelper: removeLiquidity different values of bptAmount argument and bptAmount inside requestData can steal funds from the helper contract.

OrigamiBalancerComposableStablePoolHelper is used as a helper to provide liquidity to balancer pool. There is an issue with the helper function removeLiquidity: the amount of liquidity to withdraw is given in 2 different arguments to this function:

- bptAmount argument
- bptAmount encoded in the requestData.userData argument.

The following 2 lines from the removeLiquidity function use different amounts:

```
lpToken.safeTransferFrom(msg.sender, address(this), bptAmount);
balancerVault.exitPool(poolId, address(this), recipient, requestData);
```

The first line transfers amount of liquidity given in the bptAmount argument from the user to helper contract.

The second line withdraws liquidity from the balancer pool with the liquidity amount given in the requestData.userData.

Since these 2 values are both given by the user, the mismatch can cause different amount of liquidity being taken from the user and withdrawn from the balancer pool.

Likelihood

High, the function is permissionless and such arguments can easily be crafted by any user.

Impact

Depending on which argument is greater, the following 2 impacts can happen:

- User steals all balancer pool liquidity tokens from the balancer contract (if the first argument is 0 – no tokens are taken from user, then liquidity withdrawn from balancer comes from tokens helper contracts owns) OR

- Excess unrecoverable pool liquidity tokens remain in the helper contract (if the first argument is greater than liquidity withdrawn).

Both impacts are low severity, because the helper contract is not supposed to keep any tokens, and any excess tokens will be user mistake to provide incorrect arguments.

Possible mitigation

Remove `bptAmount` argument from the `removeLiquidity` function, and decode it from the `requestData` instead.

Status: Fixed

Fix Review: `bptAmount` is now required to be equal to `bptAmount` encoded in `requestData.userData`.

5.4. Informational findings

5.4.1. [I-1] OrigamiMath.addBps unchecked addition will cause result being smaller than input amount if basisPoints is close to uint256.max.

OrigamiMath has multiple functions to make different calculations involving bps (percentage basis points). All functions except addBps provide correct results for all possible input amounts or revert. However, addBps has unchecked addition which can silently overflow and cause the incorrect result:

```
unchecked {
    numeratorBps = BASIS_POINTS_DIVISOR + basisPoints;
}

// Round up for max amounts out expected
result = mulDiv(
    inputAmount,
    numeratorBps,
    BASIS_POINTS_DIVISOR,
    roundingMode
);
```

For example, if basisPoints equals (uint256.max + 1 – BASIS_POINTS_DIVISOR), then the numeratorBps is set to 0 due to overflow, and thus result will be 0, which is incorrect.

While this function is internal and the basisPoints are expected to be set by trusted admin, or the result can be checked by the caller, it should still be clear for developers of such addBps behaviour, or be fixed to revert in such case.

Possible mitigation:

Either remove the **unchecked** keyword (to perform checked addition) and/or add comment to make it clear of the function behaviour with large basisPoints amounts.

Status: Fixed

Fix Review: Fixed by removing **unchecked**, thus it now reverts on overflows.

5.4.2. [I-2] OrigamiErc4626: areDeplositsPaused and areWithdrawalsPaused are not used anywhere and deposits/withdrawals are possible even if these functions return true.

OrigamiErc4626 implementation has 2 functions: areDepositsPaused and areWithdrawalsPaused. However, these functions are not used anywhere and do not affect the deposit or withdrawal actions.

The OrigamiBoycoVault which inherits from OrigamiErc4626 is correct, reverting deposits and withdrawals in the OrigamiBoycoUsdcManager deposit and withdraw functions if necessary.

So while the actual code in scope is correct and there are no issues, a more correct behaviour is to revert deposits and withdrawals directly in OrigamiErc4626 since there are already specific functions to pause deposits and/or withdrawals.

Possible mitigation:

Revert deposit, mint, withdraw, redeem in OrigamiErc4626 when corresponding functions (areDepositsPaused / areWithdrawalsPaused) return true.

Status: Fixed

Fix Review: Fixed as suggested.

5.4.3. [I-3] OrigamiBalancerComposableStablePoolHelper: fromInternalBalances is expected to be false by the code, but is not enforced.

OrigamiBalancerComposableStablePoolHelper is used as a helper to provide liquidity to balancer pool. The helper function addLiquidity expects the fromInternalBalances to be false, as evident from the code:

```
        for (uint256 i; i < _numTokens; ++i) {
            pullAndApproveToBalancer(IERC20(requestData.assets[i]), requestData.maxAmountsIn[i]);
        }
...
    function pullAndApproveToBalancer(IERC20 token, uint256 amount) private {
        if (amount > 0) {
            token.safeTransferFrom(msg.sender, address(this), amount);
            token.forceApprove(address(balancerVault), amount);
        }
    }
```

The code above pulls and approves tokens in full amounts. If fromInternalBalances is true, then some or all amount of the token is taken from internal balancer balances, and thus balancer transfers amounts less than maxAmountsIn from the helper, leaving the excess in the helper contract.

This can be cosidered a user mistake to use the function with fromInternalBalances == true, thus this is only informational finding. Additionally, since balancer pulls from internal balance of the caller, this means that an internal balance of the helper contract should be non-0 to cause the issue, which is highly unlikely and shouldn't happen.

Possible mitigation:

Either require fromInternalBalances to be false, or pull correct amount of tokens from the user to handle both fromInternalBalances values correctly.

Status: Fixed

Fix Review: fromInternalBalances and toInternalBalances are required to be false.

5.4.4. [I-4] OrigamiBalancerComposableStablePoolHelper: a function to recover tokens can be useful.

OrigamiBalancerComposableStablePoolHelper is used as a helper to provide liquidity to balancer pool. This smart contract is non-upgradable but it can end up with some tokens balances either as user mistakes or by some issues, a few of which are mentioned in this report. These tokens will be unrecoverable, thus it's advised to add functions to recover tokens from the helper contract.

Status: Fixed

Fix Review: Added permissioned recoverToken function.