Guardefy

Origami lovToken

Euler Borrow/Lend Integration

Audit Report

prepared by Pavel Anokhin (@panprog)

April 22, 2025

Version: 1

Confidential

Guardefy

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: 72cfaddba6a00e7c0b7b2cd766d654fef70249a8

Files in scope:

```
./contracts/common/borrowAndLend/OrigamiEulerV2BorrowAndLend.sol
./contracts/investments/lovToken/OrigamiLovToken.sol
./contracts/investments/lovToken/managers/OrigamiAbstractLovTokenManager.sol
./contracts/investments/lovToken/managers/OrigamiLovTokenMorphoManager.sol
```

4. Audit Timeline

Audit start: April 14, 2025

Audit report delivered: April 22, 2025

Fixes reviewed:

5. Findings

5.1. High severity findings

None found.

5.2. Medium severity findings

5.2.1. [M-1] `OrigamiLovTokenMorphoManager._rebalanceUp` reverts if any slippage occurs when swapping base token to quote token in `OrigamiEulerV2BorrowAndLend`.

When doing rebalance up (reducing leverage), `OrigamiLovTokenMorphoManager` assumes that the repayment amount matches user specified amount exactly (except if forced):

```
        if (_debtRepaidAmount != params.repayAmount) {
            if (!force) revert CommonEventsAndErrors.InvalidAmount(address(_debtToken),
params.repayAmount);
        }
```

While Morpho's BorrowAndLend handles any excess of repayment amount separately as a "repay surplus", Euler's BorrowAndLend simply swaps base token to quote token and repays everything that was swapped, with `params.repayAmount` serving as a minimum repayment amount for slippage control:

```
        uint256 borrowTokensReceived = swapper.execute(_supplyToken, withdrawn,
_borrowToken, swapData);
        if (borrowTokensReceived < minBorrowTokensReceived) {
            revert CommonEventsAndErrors.Slippage(minBorrowTokensReceived,
borrowTokensReceived);
        }

        // Within _repay(), if (borrowTokensReceived > outstanding debt), only outstanding debt is
repaid
        debtRepaidAmount = _repay(borrowTokensReceived);
```

This means that non-forced `_rebalanceUp` will almost always revert for Euler vaults, because the repayAmount will be less than actual repayment amount most of the time since it's the low bound for slippage control, not the expected repayment amount.

Likelihood

High. The issue happens almost always when `OrigamiLovTokenMorphoManager.rebalanceUp` is called.

Impact

Most `rebalanceUp` calls will revert.

Possible mitigation

Either fix the Euler's BorrowAndLend to match the other adapters, handling any excess repayment as surplus repayment amount. Alternatively, just remove (or change to `<`) the repayAmount check in `OrigamiLovTokenMorphoManager._rebalanceUp` since it's verified downstream in BorrowAndLend anyway.

Status: Reported

Fix Review:

5.3. Low severity findings

5.3.1. [L-1] OrigamiEulerV2BorrowAndLend._supply assumes that amount sent to the vault equals the amount credited which might not be the case.

When OrigamiLovToken.investWithToken is called by the user, user is minted lovToken shares proportional to amount of base token sent by the user relative to vault's redeemable assets:

```
    function _supply(uint256 supplyAmount) internal returns (uint256 suppliedAmount) {
        if (supplyAmount < type(uint256).max) {
            supplyVault.deposit(supplyAmount, address(this));
            suppliedAmount = supplyAmount;
        } else {
            // when supplyAmount is type(uint256).max, the EVault will attempt to deposit all assets held
by the
            // depositor and will revert if hitting the vault's supply cap
            // This is preferred over supplying up to the cap, as it can affect the A/L ratio negatively
            uint256 balanceBefore = _supplyToken.balanceOf(address(this));
            supplyVault.deposit(supplyAmount, address(this));
            suppliedAmount = balanceBefore - _supplyToken.balanceOf(address(this));
        }
    }
```

The issue is that after the deposit into the underlying Euler vault, the actual amount redeemable by lovToken can be a different (smaller) amount due to rounding and possible deposit or exit fees (charged by the Euler vault). For example:

- lovToken redeemable assets = 100, totalSupply = 100
- user deposits 100 assets and is minted 100 lovToken shares
- however, if Euler vault charges 1% deposit fee, the actual lovToken redeemable amount after the deposit will be 199 assets
- this means that 100 shares are now worth 99.5 assets: existing lovToken users lose 0.5 assets total, while the deposited user is unfairly credited with 99.5 assets (instead of 99 assets).

Note, that current reference Euler vault implementation doesn't have deposit or exit fees, however Euler vaults are very flexible and can implement everything, it's also possible that the reference implementation is upgraded in the future as it's upgradable.

Likelihood

Rounding issue is likely (user deposits 100000 assets, but Euler vault rounds it down to 99999 assets credited to lovToken), but is at most dust (1 wei) loss.

Euler vault deposit fees is a possible future integration issue which might not require immediate fix.

Impact

Low for dust loss due to rounding, informational for possible future Euler vault fees.

Possible mitigation

Calculate actual lovToken's redeemable amount change after the deposit and mint lovToken shares off this amount, rather than amount sent by the user.

Status: Reported.

Fix Review:

5.4. Informational severity findings

5.4.1. [I-1] Spot price oracle used to evaluate user AL range should match Euler vault's price oracle.

When users deposit to or withdraw from the lovToken, the actual lovToken's leverage changes accordingly (decreases with deposits, increases with withdrawals). The `OrigamiAbstractLovTokenManager` limits deposits and withdrawals such that actual leverage must not move outside the `userALRange` (it can be broken by external market price move, but user deposits and withdrawals must not make it worse).

When the lovToken's actual leverage is calculated, the manager uses spot price oracle to convert quote token amount into corresponding base token amount.

However, Euler vault itself also evaluates the lovToken's leverage to ensure the account is healthy using its own quote asset price oracle.

If these 2 oracles use different sources to evaluate prices, this can lead to potential problems such as:

- User unable to deposit or withdraw when user AL range is correct, but lovToken position in Euler vault is unhealthy (for example, user uses `maxExit` amount to redeem, but it reverts as the amount is actually incorrect because it breaks Euler vault max leverage)
- User can intentionally redeem max amount which will bring the lovToken position in Euler vault very close to liquidation, even if the `userALRange` is not close to max Euler vault leverage.

Most of the time this can be solved with some safety built into `userALRange`, so that it's not close to Euler vault max leverage. However, in extreme situations, especially when the asset price depegs significantly, the price can be very different in different oracle sources / exchanges and it can lead to unpredictable issues and increased risk of liquidation.

Possible mitigation

Ideally, Euler vault and lovToken manager should use the same oracle source for assets spot prices. If this is not possible, some sanity check on difference between euler oracle and lovToken manager oracle should be made, to at least ensure lovToken's Euler position is never above certain max leverage regardless of `userALRange` check. This should at least cover the most dangerous case where lovToken position might become too risky / liquidated.

Status: Reported