Guardefy

Mono Cooler

Audit Report

prepared by Pavel Anokhin (@panprog)

March 9, 2025

Version: 2

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/OlympusDAO/olympus-v3

commit hash: 7a599fd50ce377714856102fde11af73c5f4f6b6

Files in scope:

```
./src/external/cooler/DelegateEscrow.sol
./src/external/cooler/DelegateEscrowFactory.sol
./src/libraries/CompoundedInterest.sol
./src/libraries/SafeCast.sol
./src/modules/DLGTE/DLGTE.v1.sol
./src/modules/DLGTE/OlympusGovDelegation.sol
./src/policies/cooler/CoolerLtvOracle.sol
./src/policies/cooler/CoolerTreasuryBorrower.sol
./src/policies/cooler/MonoCooler.sol
./src/policies/utils/PolicyAdmin.sol
./src/policies/utils/PolicyEnabler.sol
./src/policies/utils/RoleDefinitions.sol
```

4. Audit Timeline

Audit start: February 13, 2025

Audit report delivered: February 23, 2025

Fixes reviewed: March 9, 2025

5. Findings

5.1. High severity findings

None found.

5.2. Medium severity findings

5.2.1. [M-1] MonoCooler liquidations transactions can be reverted by front-running with random delegations or undelegations.

When account borrowing from MonoCooler becomes insolvent (liquidation LTV is breached), it can be liquidated by anyone. All account collateral is seized and part of it is paid as a liquidation incentive to liquidator.

Since users can delegate their MonoCooler collateral, delegated account balance can not be withdrawn immediately and has to be undelegated first. This means that all delegated balance must be undelegated before the liquidation. When account is unhealthy, undelegation can be done by anyone either in batchLiquidate directly, or in applyUnhealthyDelegations (which is needed if there are too many delegations and there is not enough gas in a single transaction to undelegate all).

The issue is that the account owner can delegate or undelegate at any time without any limitations and thus can front-run any liquidation attempt by randomly delegating tiny amounts. Since any liquidation attempt must provide the list of addresses and exact amounts to undelegate, any change in delegations (either increase or decrease of delegated amounts) will revert the liquidation transaction: either not all collateral will be undelegated and thus can't be burned, or too much is attempted to be undelegated.

The attacker can borrow huge amounts, then intentionally let his account become liquidatable, and then keep front-running all attempts to liquidate the account with random delegations until the incentive to liquidate account is close to 100% of collateral, at which point the attacker can liquidate the account himself, thus getting all collateral back as well as keeping all loaned amount (essentially stealing it).

This are the possible ways for the attacker to front-run liquidation transaction to revert it:

- applyDelegations (delegate or undelegate random amounts)
- addCollateral (add random tiny amount of collateral and delegate or undelegate random amounts)
- applyUnhealthyDelegations (undelegate random amounts) – can even be called by any user, not just the attacker.
- Use the other policy which allows user to change delegations and either delegate or undelegated random amounts, or add tiny amount of collateral via another policy, this will revert liquidation in the following line:

```
// Cannot undelegate more collateral than required in order to fullfill a liquidation.
if (undelegatedBalance > acctCollateral) revert InvalidDelegationRequests();
```

Likelihood

Medium. It can take a large amount of time for the account to be liquidatable and even more time for the liquidation incentive to be large enough for the attack to be profitable, all this time attacker will have to keep front-running all liquidation attempts. There is some

probability that he will be unable to front-run all transaction and some liquidation might still come through. Still, this is a serious denial of service for the very imporant liquidation mechanism.

Impact

In the worse case all balance of debt token can be stolen by the attacker from the Olympus Treasury.

Possible mitigation

Instead of using user-provided list of undelegations during liquidation, undelegate all by iterating all delegated accounts and undelegating full balance (up to required amount of collateral). This will make the front-running with delegations useless. In case there is not enough gas to undelegate everything, `applyUnhealthyDelegations` can still be used permissionlessly, however, it can also be front-run to be reverted, so it's better to make it also undelegated all without the list of undelegations, and take a parameter of max number of accounts to undelegated (so that it can be broken out into several transactions if there is not enough gas to undelegated all in 1 transaction).

Status: Fixed

Fix Review: Fixed as suggested, undelegations during liquidations and unhealthy undelegations now auto-undelegate by iterating all delegates until necessary amount is reached.

5.3. Low severity findings

5.3.1. [L-1] `OlympusGovDelegation.accountDelegationsList`: incorrect `maxPossibleEndIndex` calculation.

There is an incorrect calculation of maxPossibleEndIndex:

```
// end index is the max of the requested items or the length
uint256 requestedEndIndex = startIndex + maxItems - 1;
uint256 maxPossibleEndIndex = length - startIndex - 1;
if (maxPossibleEndIndex < requestedEndIndex) requestedEndIndex = maxPossibleEndIndex;
```

As this is the index of the last item of the list, it should be simply `length – 1` (`length – startIndex – 1` is the length of the list, not the end index).

Example: if `length = 10`, then `accountDelegationsList(account, 2, 10)` will return items in the range `[2..(10-2-1)] = [2..7]` instead of `[2..9]`.

Likelihood

High, most requests where `startIndex` is not 0 will return incorrect number of items.

Impact

No direct impact in the code in scope, but integrations using this function to fetch delegations will have incorrect lists returned to them.

Possible mitigation

```
maxPossibleEndIndex = length - 1;
```

Status: Fixed

Fix Review: Fixed as suggested. Additionally, another issue with incorrect indexing inside the results array was fixed.

5.3.2. [L-2] Very late liquidations in MonoCooler will have liquidation incentive higher than account collateral and will revert.

The liquidation incentive for liquidating unhealthy accounts in MonoCooler is calculated as:

```
// The incentive is calaculated as the excess debt above the LLTV, in collateral terms
// excessDebt [gOHM] = currentDebt [USDS] / LLTV [USDS/gOHM] - collateral [gOHM]
status.currentIncentive = (uint256(status.currentDebt).divWadUp(
    gStateCache.liquidationLtv
) - status.collateral).encodeUInt128();
```

When liquidation is very late for any reason, the debt might grow so high that the liquidation incentive becomes higher than collateral, for example:

- Debt = 24
- Collateral = 9
- LLTV = 1.2
- Liquidation Incentive = 24 / 1.2 – 9 = 11

When trying to liquidate such account, it will revert in the following line:

```
uint128 gOhmToBurn = totalCollateralClaimed - totalLiquidationIncentive;
```

Likelihood

Low since liquidation itself is supposed to be a very rare event (with the debt interest being 0.5% annually at the start and 1% difference between liquidation and origination LTV, account can become liquidatable only after 2 years), and incentive will grow very slowly, it might take hundreds of years to reach this level.

Impact

User account can not be liquidated normally (can still be liquidated together with the other accounts or anyone can repay/add collateral to still liquidate it – still this will be a loss of funds to fix the issue).

Possible mitigation

Limit liquidation incentive to account collateral.

Status: Fixed

Fix Review: Fixed as suggested, max incentive is limited by account collateral.