decurity

Smart Contract Security Audit Report

Gearbox Integrations Update

1. Contents

2. General Information

This report contains information about the results of the security audit of the Gearbox (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 03/06/2025 to 10/06/2025.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repositories: integrations-v3, periphery-v3, and oracles-v3.

The initial review was done for the commits 08e810, be2d3a, and 7553a8 accordingly. The final review was done for commits 0cba62, 44cfb9, and 7553a8.

The following contracts have been tested:

integrations-v3:

- contracts/adapters/curve/CurveV1_2.sol
- contracts/adapters/curve/CurveV1_3.sol
- contracts/adapters/curve/CurveV1_4.sol
- contracts/adapters/curve/CurveV1_Base.sol
- contracts/adapters/curve/CurveV1_DepositZap.sol
- contracts/adapters/curve/CurveV1_StableNG.sol
- contracts/adapters/curve/CurveV1_stETH.sol
- contracts/adapters/erc4626/ERC4626Adapter.sol

- contracts/adapters/erc4626/ERC4626ReferralAdapter.sol
- contracts/adapters/fluid/FluidDexAdapter.sol
- contracts/adapters/infrared/InfraredVaultAdapter.sol
- contracts/adapters/mellow/Mellow4626VaultAdapter.sol
- contracts/adapters/sky/StakingRewardsAdapter.sol
- contracts/adapters/traderjoe/TraderJoeRouterAdapter.sol
- contracts/helpers/balancer/BalancerV3RouterGateway.sol
- contracts/helpers/convex/ConvexV1_StakedPositionToken.sol
- contracts/helpers/curve/CurveV1_stETHGateway.sol
- contracts/helpers/fluid/FluidDexETHGateway.sol
- contracts/helpers/infrared/InfraredVaultPhantomToken.sol
- contracts/helpers/lido/LidoV1_WETHGateway.sol
- contracts/helpers/mellow/MellowWithdrawalPhantomToken.sol
- contracts/helpers/sky/StakingRewardsPhantomToken.sol

periphery-v3:

- contracts/emergency/MultiPause.sol
- contracts/emergency/TreasuryLiquidator.sol

oracles-v3:

- contracts/oracles/ConstantPriceFeed.sol
- contracts/oracles/curve/CurveTWAPPriceFeed.sol

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,

- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided "as is" and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer's project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered two medium security issue. The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement). The Gearbox team has given the feedback for the suggested changes and explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of June 16, 2025.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Lack of wrappedUnderlying validation	periphery-v3/contracts/emergency/TreasuryLiquidator.sol	Medium	Fixed
Mellow protocol inconsistent withdrawals	integrations-v3/contracts/adapters/mellow/Mellow4626VaultAdapter.sol	Medium	Fixed
Assignment instead of revert in case of upperBound achievement	oracles-v3/contracts/oracles/curve/CurveTWAPPriceFeed.sol	Low	Acknowledged
Using named parameters in mapping is best practice	periphery-v3/contracts/emergency/TreasuryLiquidator.sol	Info	Fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Lack of wrappedUnderlying validation

Risk Level: **Medium**

Status: Fixed in the commit.

Contracts:

- periphery-v3/contracts/emergency/TreasuryLiquidator.sol

Location: Function: `partiallyLiquidateFromTreasury`.

Description:

The `partiallyLiquidateFromTreasury()` function accepts `wrappedUnderlying` address that may be used to withdraw assets from the pool.

Unfortunately, TreasuryLiquidator contract does not perform a validation to ensure that the asset of the `wrappedUnderlying` matches the actual `underlyingToken` of the CreditFacade.

The TreasuryLiquidator has the necessary approvals to transfer a range of assets. This introduces a potential edge case:

- A scenario can arise where the underlying token of a given CreditFacade is a stablecoin, while its corresponding `wrappedUnderlying` points to a different asset (e.g., BTC).
- If someone pre-funds the liquidator with the required amount of stablecoins, liquidation could proceed despite the mismatch. However, the BTC sent to the liquidator as a result would remain locked until a reverse operation is executed (i.e., where `wrappedUnderlying` points to the stablecoin and underlying to BTC).

Remediation:

Consider adding a check:

```
if (IPoolV3(wrappedUnderlying).underlyingToken() != underlying) revert
WrongWrappedUnderlyuing();
```

5.2. Mellow protocol inconsistent withdrawals

Risk Level: Medium

Status: Fixed in the commit.

Contracts:

- integrations-v3/contracts/adapters/mellow/Mellow4626VaultAdapter.sol

Location: Function: `claim, withdrawPhantomToken`.

Description:

The Mellow vault `claim` function reduces `maxAmount` to the actually claimable amount, which results in inconsistencies during claims. Even if the provided amount is larger than the claimable amount, the function will not revert. This behaviour violates integrity and may lead to unpredictable outcomes in other parts of the protocol.

Remediation:

Consider checking that claimed amount equals to the provided amount.

5.3. Assignment instead of revert in case of upperBound achievement

Risk Level: Low

Status: We do not do a revert on purpose if the price has exceeded the limit, because it is usually not dangerous for users, but allows the protocol to continue to function. At the lower limit, a revert is necessary, because otherwise the oracle will overstate the price, and usually breaking the lower limit is something urgent, and it is better to revert in such cases.

Contracts:

- oracles-v3/contracts/oracles/curve/CurveTWAPPriceFeed.sol

Location: Function: `latestRoundData`.

Description:

If the upperBound is reached, the `latestRoundData` function assigns upperBound to `exchangeRate` instead of reverting:

```
if (exchangeRate > upperBound) exchangeRate = upperBound;
```

This logic may lead to unexpected behavior and potential loss of user funds.

Remediation:

Consider reverting in case upperBound is reached.

5.4. Using named parameters in mapping is best practice

Risk Level: Info

Status: Fixed in the commit.

Contracts:

- periphery-v3/contracts/emergency/TreasuryLiquidator.sol

Description:

Using named parameters in mapping is considered best practice for code clarity and maintainability.

There are 2 instance of this issue:

```
File: contracts/emergency/TreasuryLiquidator.sol

42:     mapping(address => bool) public isLiquidator;

46:     mapping(address => mapping(address => uint256)) public
minExchangeRates;
```

Remediation:

Use named parameters in mapping declarations for better readability.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.