PUBLIC

# Code Assessment

## of the Gearbox V3 Integrations Smart Contracts

November 11, 2023

Produced for

by

# Contents

# 1 Executive Summary

Dear Gearbox Team,

Thank you for trusting us to help Gearbox Protocol with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Gearbox V3 Integrations according to Scope to support you in forming an opinion on their security risks.

Gearbox Protocol refactors the adapter contracts used to interact with third-party protocols.

The most critical subjects covered in our audit are the functional correctness of the contracts, the adapter configuration, the movement of the assets, and the interaction with the rest of the Gearbox system. A high severity issue was uncovered in one of the iterations where anyone could redeem on behalf of any user by front-runnng the signed permit or back-running the approval of the user. The issues have been addressed in the final commit. All in all, all the issues reported have been addressed. Security regarding all the aforementioned subjects is high.

The general subjects covered are access control, documentation and specification, gas efficiency, and the complexity of the implementation. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security as no more issues were uncovered. We need to emphasize that the interactions between different components of the Gearbox system are complex. Moreover, the contracts in this scope have undergone many changes during the review. This in combination with the fact that the reviews are limited in time reduces our confidence in the assessment of the system's security level.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

## 1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 2 |
| • Code Corrected | 2 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 2 |
| • Code Corrected | 1 |
| • Acknowledged | 1 |

# 2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

## 2.1 Scope

The assessment was performed on the source code files inside the `contracts/` folder of the Gearbox V3 Integrations repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

| V | Date | Commit Hash | Note |
|---|---|---|---|
| 1 | 25 Sept 2023 | dddd06719bda2eee3f47d44cf182039f002c12f8 | Initial Version |
| 2 | 30 Oct 2023 | 5036094502ffe70b433ac7c89edce6990ace5ed2 | Added <x>_diff functions |
| 3 | 6 Nov 2023 | 66bc3fd399189fa6fe73579bb8666f6e416302cf | Inlining of internal <x>Diff functions |
| 4 | 8 Nov 2023 | 74888bcf007aab373d0504cfed5ca78c8d8f865e | Added pool migration zapper |
| 5 | 11 Nov 2023 | 302c635e67c0017f5f7d91d9c4c56199c624c4f6 | Fix of redemption mechanism |

For the solidity smart contracts, the compiler version `0.8.17` was chosen.

The scope of this review is limited to the changes in the following files and folders compared to the last commits of the Gearbox V2.1 report.

The previous commit for the `integrations-v3` repository is `02f239fee250fb11b16a28974e71e73264de50b2`.

The following contracts are in the scope of the review:

```
adapters:
    AbstractAdapter.sol
    aave:
        AaveV2_LendingPoolAdapter.sol
        AaveV2_WrappedATokenAdapter.sol
    balancer:
        BalancerV2VaultAdapter.sol
    compound:
        CompoundV2_CErc20Adapter.sol
        CompoundV2_CEtherAdapter.sol
        CompoundV2_CTokenAdapter.sol
    convex:
        ConvexV1_BaseRewardPool.sol
        ConvexV1_Booster.sol
    curve:
        CurveV1_2.sol
        CurveV1_3.sol
        CurveV1_4.sol
```

```
        CurveV1_Base.sol
        CurveV1_DepositZap.sol
        CurveV1_stETH.sol
    erc4626:
        ERC4626Adapter.sol
    lido:
        LidoV1.sol
        WstETHV1.sol
    uniswap:
        UniswapV2.sol
        UniswapV3.sol
    yearn:
        YearnV2.sol
helpers:
    aave:
        AaveV2_WrappedAToken.sol
    compound:
        CompoundV2_CEtherGateway.sol
    convex:
        ConvexV1_StakedPositionToken.sol
    curve:
        CurveV1_stETHGateway.sol
    lido:
        LidoV1_WETHGateway.sol
integrations:
    TokenType.sol
    aave:
        DataTypes.sol
        IAToken.sol
        ILendingPool.sol
    balancer:
        IAsset.sol
        IBalancerQueries.sol
        IBalancerStablePool.sol
        IBalancerV2Vault.sol
        IBalancerWeightedPool.sol
    compound:
        ICErc20.sol
        ICEther.sol
        ICToken.sol
    convex:
        IBaseRewardPool.sol
        IBooster.sol
        IConvexToken.sol
        IRewards.sol
        Interfaces.sol
    curve:
        ICRVToken.sol
        ICurvePool.sol
        ICurvePoolStETH.sol
        ICurvePool_2.sol
        ICurvePool_3.sol
        ICurvePool_4.sol
        ICurveRegistry.sol
    lido:
```

```
        IstETH.sol
        IwstETH.sol
    uniswap:
        BytesLib.sol
        IQuoter.sol
        IUniswapV2Router01.sol
        IUniswapV2Router02.sol
        IUniswapV3.sol
        IUniswapV3SwapCallback.sol
        Path.sol
    yearn:
        IYVault.sol
interfaces:
    aave:
        IAaveV2_LendingPoolAdapter.sol
        IAaveV2_WrappedATokenAdapter.sol
    balancer:
        IBalancerV2VaultAdapter.sol
    compound:
        ICompoundV2_CTokenAdapter.sol
    convex:
        IConvexV1BaseRewardPoolAdapter.sol
        IConvexV1BoosterAdapter.sol
    curve:
        ICurveV1Adapter.sol
        ICurveV1_2AssetsAdapter.sol
        ICurveV1_3AssetsAdapter.sol
        ICurveV1_4AssetsAdapter.sol
    erc4626:
        IERC4626Adapter.sol
    lido:
        ILidoV1Adapter.sol
        IwstETHV1Adapter.sol
    uniswap:
        IUniswapV2Adapter.sol
        IUniswapV3Adapter.sol
    yearn:
        IYearnV2Adapter.sol
    zappers:
        IERC20ZapperDeposits.sol
        IETHZapperDeposits.sol
        IZapper.sol
zappers:
    DTokenDepositZapper.sol
    DTokenFarmingZapper.sol
    ERC20ZapperBase.sol
    ETHZapperBase.sol
    UnderlyingDepositZapper.sol
    UnderlyingFarmingZapper.sol
    WATokenDepositZapper.sol
    WATokenFarmingZapper.sol
    WETHDepositZapper.sol
    WETHFarmingZapper.sol
    WstETHDepositZapper.sol
    WstETHFarmingZapper.sol
```

```
ZapperBase.sol
traits:
    DTokenTrait.sol
    DepositTrait.sol
    FarmingTrait.sol
    UnderlyingTrait.sol
    WATokenTrait.sol
    WETHTrait.sol
    WstETHTrait.sol
```

### 2.1.1 Excluded from scope

Any contracts not explicitly listed above are out of the scope of this review. Third-party libraries are out of the scope of this review. More specifically, the contracts with which these adapters interact are assumed safe and to work as expected.

## 2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Gearbox Protocol integrations that facilitate the interaction of Gearbox with external protocols. These integrations are refactored versions of the respective contracts used in previous versions of the protocol. Furthermore, some Zapper contracts are introduced to aggregate common actions for users who want to lend funds to Gearbox pools.

### 2.2.1 Adapters

Adapters facilitate interaction with third-party protocols using the `CreditAccounts`'s funds. All adapters follow the same interaction paradigm, i.e., interactions with the Credit Account must be done through the Credit Facade. Generally, adapters implement the very same function interfaces as the target contract. Unsupported functions are no longer present. Some adapters implement variations of functions ending with `..all()`. These functions spend the whole balance the `CreditAccount` has of the spent token. For Version 3 most adapters remain unchanged from previous versions with the main difference being that they return the tokens to be enabled or disabled to their caller. The reader can refer to the report of version 2.1. More specifically there are adapters for the following protocols: Yearn V2, Uniswap V2, Uniswap V3, Balancer V2, Compound V2, Curve, Lido, Convex. Finally, a new generic adapter for ERC4626 was added.

#### 2.2.1.1 ERC4626

The interface supported by the adapter is the following:

- `deposit(assets, receiver)`: deposits the amount of assets and enables the token that represents the shares of the vault. The `receiver` is ignored as it's always the credit account. It enables the shares token.
- `depositAll()`: deposits the whole balance of the underlying asset held by the credit account and enables the token that represents the shares of the vault. It disables the underlying token and enables the shares token.

- `mint(shares, receiver)`: deposits the appropriate amount to mint a specified amount of shares. The `receiver` is ignored as it's always the credit account. It enables the shares token.
- `withdraw(assets, receiver, owner)`: redeems the appropriate amount of shares to withdraw the amount of the underlying assets specified by `assets`. The rest of the variables are ignored as they are always the credit account. It enables the underlying token.
- `redeem(shares, receiver, owner)`: redeem `shares` amount of shares for some assets. The rest of the variables are ignored as they are always the credit account. It enables the underlying token.
- `redeemAll()`: It redeems the whole amount of shares held by the credit account. It disables the shares token and enables the underlying token.

### 2.2.2 Zappers:

All the currently available zappers implement a similar functionality and they share the same interface:

- `deposit[WithReferral]()`: allows users to wrap their tokens and lend them using optionally a referral to a Gearbox lending pool.
- `redeem[WithPermit]()`: allows users to redeem their wrapped tokens from a pool and immediately unwrap them. A user can redeem the tokens on behalf of another user through a permission mechanism.
- `previewDeposit()`: a view function which returns the number of shares users will receive if they deposit an `amount` of the underlying token.
- `previewRedeem()`: a view function that returns the amount of unwrapped tokens a user will receive if they redeem an amount of `shares` of the pool.

More specifically:

- `WATokenZapper`: wraps ATokens to WATokens. For this, it interacts with the `AaveV2_WrappedToken` contract.
- `WstETHZapper`: wraps stETH to WstETH.
- `WETHZapper`: wraps ETH to WETH.

## 2.3 Changes in Version 2

- The adapters have new functionality that allows callers to specify the leftover amount of a specific token balance in the CreditAccount. These functions have the form `<x>Diff`.
- The architecture of the Zappers has been updated and allows more functionality than before. All the zappers implement either `ERC20ZapperBase` or `ETHZapperBase`, which both inherit from `ZapperBase`. The zappers are built on top of these with a combination of some of the following traits:
    - `DepositTrait`: implements functionality to deposit some token in a pool
    - `FarmingTrait`: implements functionality to deposit pool LP tokens in a 1inch farming pool
    - `UnderlyingTrait`: implements functionality to deal directly with the underlying token
    - `WATokenTrait`: implements functionality to wrap/unwrap `aToken/WAToken`
    - `WETHTrait`: implements functionality to wrap/unwrap `ETH/WETH`
    - `WstETHTrait`: implements functionality to wrap/unwrap `stETH/WstETH`

The composition of the traits allows efficient wrap/unwrap/deposit/withdrawal from/to Gearbox liquidity and 1inch farming pools in one transaction for liquidity providers. The available zappers are:

- `Underlying[Deposit|Farming]Zapper`: allows users to add liquidity to a pool in one transaction by depositing the underlying with a (signed) permit. If the farming version is used, the LP token will be deposited in 1inch farming pool. Users can also redeem their shares in the underlying token.
- `WAToken[Deposit|Farming]Zapper`: allows users to directly add liquidity to a pool using their `aTokens` by wrapping them to `waToken`. If the farming version is used, the LP token will be deposited in 1inch farming pool. Users can also redeem their shares in `aToken`.
- `WETH[Deposit|Farming]Zapper`: allows users to add liquidity in one transaction to a pool with `WETH` as underlying by wrapping `ETH` to `WETH`. If the farming version is used, the LP token will be deposited in 1inch farming pool. Users can also redeem their shares in `ETH`.
- `WstETH[Deposit|Farming]Zapper`: allows users to add liquidity to a pool with `WstETH`, by converting `stETH` to `WstETH` for use within the pool. If the farming version is used, the LP token will be deposited in 1inch farming pool. Users can also redeem their shares in `stETH`.

## 2.4 Changes in Version 3

The functions `<x>All` have been removed as the same result can easily be achieved with the `<x>Diff` functions.

## 2.5 Changes in Version 4

A new zapper trait for pool LP tokens migration has been added:

- `DTokenTrait`: implements functionality to withdraw liquidity from an old pool

The new zappers are:

- `DToken[Deposit|Farming]Zapper`: allows users to migrate their liquidity from an old pool used in previous versions of the protocol (V1, V2, V2.1) to a new one (used in V3). If the farming version is used, the LP token will be deposited in 1inch farming pool. The zapper allows the migration to happen only in one direction.

# 3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

# 4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

| Likelihood | Impact | | |
|---|---|---|---|
| | High | Medium | Low |
| High | Critical | High | Medium |
| Medium | High | Medium | Low |
| Low | Medium | Low | Low |

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

# 5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 1 |



## 5.1 `WrappedAToken` Does Not Implement Its Interface

Design Low Version 1 Acknowledged

*CS-GEARV3INTGRTNS-001*

The interface `IWAToken` is defined in `oracles-v3/contracts/interfaces/aave/IWAToken.sol` and used to interact with `WrappedAToken`, but `WrappedAToken` does not implement fully this interface.

---

**Acknowledged:**

Gearbox Protocol responded:

```
The contract conforms to the interface, which is sufficient.(Note, however, that external interfaces
in oracles are reduced to only have the functions necessary in price feeds).
```

# 6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 2 |

- Back-running Redemption Approvals Code Corrected
- Front-running the Redeem Code Corrected

| | |
|---|---|
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 1 |

- Number of Underlying Tokens in Metapools Code Corrected

| | |
|---|---|
| Informational Findings | 1 |

- Gas Optimizations Code Corrected

## 6.1 Back-running Redemption Approvals

Security High Version 2 Code Corrected

*CS-GEARV3INTGRTNS-004*

When redeeming assets a user calls `redeem` to the relevant zapper. They should give approval to the zapper to be able to redeem the assets. An attacker who sees the approval can the front-run the actual redemption redeem the assets of the user.

*The issue was reported by the client during the review after an independent assessment of the codebase.*

---

**Code corrected:**

Only the `msg.sender` can use their approvals.

## 6.2 Front-running the Redeem

Security High Version 1 Code Corrected

*CS-GEARV3INTGRTNS-005*

When redeeming a token with permit, a user specifies the receiver of the redeemed assets and submits a signature which is verified as follows:

```
try IERC20Permit(tokenOut()).permit(owner, address(this), tokenOutAmount, deadline, v, r, s) {} catch {} // U:[ZB-5]
```

Note that the signature verified is not connected to the `msg.sender`. Thus, an attacker who observes the mempool can front-run and submit the same signature. Since the receiver is freely set, an attacker can redeem the assets of a user.

*The issue was reported by the client during the review after an independent assessment of the codebase.*

---

**Code corrected:**

In the current implementation, only a signature belonging to the msg.sender can be verified.

## 6.3 Number of Underlying Tokens in Metapools

Correctness | Low | Version 1 | Code Corrected

*CS-GEARV3INTGRTNS-003*

The current implementation of `CurveV1AdapterBase` assumes the metapool to be a tricrypto pool, or at least have 3 underlying tokens. If the metapool has less than 3 underlying tokens, then the constructor will revert.

---

**Code corrected:**

The function `_getCoin` will not revert if `CurvePool.coin()` reverts.

## 6.4 Gas Optimizations

Informational | Version 1 | Code Corrected

*CS-GEARV3INTGRTNS-002*

In the constructor of `CurveV1AdapterBase`, when the underlying tokens are queried for lending pools, the loop can break in the case `!success` as the calls to `underlying_coins` in following iterations will fail as well.

---

**Code corrected:**

The loop now breaks the first time `success` is false.

# 7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

## 7.1 Zappers Do Not Support Tokens With Fees

Note Version 1

Even though the pools can handle tokens with fees (USDT for now), the zappers do not support tokens with fees. If the fees on USDT were to be activated, the `Underlying[Deposit|Farming]Zapper` will stop working and users will have to deposit and withdraw manually.