PRIVATE

Code Assessment

of the Gearbox V2.1 Smart Contracts

August 2, 2023

Produced for

by

Contents

1 Executive Summary

Dear all,

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 V2.1 according to Scope to support you in forming an opinion on their security risks.

Gearbox Protocol has implemented version 2.1, an improved iteration of the existing v2 protocol. Based on lessons learned since the launch of v2, numerous enhancements and fixes have been incorporated to strengthen security, such as minimizing the attack surface. Access has been further restricted, with direct interaction with adapters no longer permitted. All interactions must now go through the CreditFacade. Additionally, new adapters have been introduced to enable credit accounts to interact with Balancer, Compound, and Aave V2, along with the addition of three new price feeds.

Our audit's most critical focus areas include verifying the proper behavior, security, and financial stability of the protocol. A significant portion of our review concentrates on ensuring the accuracy of adapters when interacting with external systems. We also examined the newly added price feeds.

Security regarding all the aforementioned subjects is high.

We also examined the code's correctness with respect to the available specification and the consistency of the implementation.

In summary, we find that the codebase of the protocol provides a high level of security.

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	1
• Code Corrected	1
Medium-Severity Findings	2
• Code Corrected	2
Low-Severity Findings	7
• Code Corrected	5
• Risk Accepted	2

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 Gearbox V2.1 repository based on the documentation files: The table below indicates the code versions relevant to this report and when they were received.

core-v2

V	Date	Commit Hash	Note
1	20 Mar 2023	529dbfb877b43af45a21dccdd443c1e23c848a64	Initial Version (V1)
2	24 Mar 2023	344687b96eb88dd71662914f1bca0374047ee478	Updated Version - total debt (V2)
3	02 Apr 2023	d5e06f3d3bb068c82df65999fa963203a30d34a9	Updated Version - optimizations (V3)
4	18 Apr 2023	2f01dcaa2512a4f51157bacce45544c51e5033b3	Version with fixes (V4)

integrations-v2

V	Date	Commit Hash	Note
1	20 Mar 2023	e0d628447c3916f70d34a033e5571b730c88574f	Initial Version (V1)

integrations-v3

V	Date	Commit Hash	Note
1	17 Apr 2023	fce737341524d9fb884c7ee29f8ac3d8a54699a2	Version with fixes (V4)
2	21 Apr 2023	e34cfbe9fb7e3b41121acce1911c4484ca60e211	Updated Balancer Adapter (V5)
3	31 Jul 2023	bb68f9a2461c1a6aa7a5c9b42b88a73de1b9d060	Updated Curve, Convex and, Uniswap Adapters (V6)
4	31 Jul 2023	02f239fee250fb11b16a28974e71e73264de50b2	Fixes for Curve and Convex Adapters (V7)

For the solidity smart contracts in `core-v2`, the compiler version `0.8.10` was chosen.

For the solidity smart contracts in `integrations-v2` and `integrations-v3`, 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 report.

The previous commit for the *core-v2* repository is *c6ca919d46dcd82fa69c89316d9ff969e89bd3f6*.

The previous commit for the *integrations-v2* repository is *c7290c3ef917f456653e7d5151dc610f338a0805*.

`core-v2`:

```
adapters/*
core/*
credit/*
libraries/*
multicall/*
oracles/*
pool/*
support/BlackListHelper.sol
tokens/*
```

`integrations-v2/integrations-v3`:

```
adapters/balancer/*
adapters/convex/* except ConvexV1_ClaimZap.sol
adapters/curve/*
adapters/lido/*
adapters/uniswap/*
adapters/yearn/*
oracles/curve/CurveCryptoLPPriceFeed.sol
```

The review of Gearbox core-v3 will be conducted at a later time. As a result, at this time the adapters in `integrations-v3` have been reviewed by taking into account only the `AbstractAdapter` from `core-v2`. Specifics related to core-v3 can only be covered after the corresponding review has been completed.

The issues related to `integrations-v2` have been fixed in the updated version of `integrations-v3` (in the `legacy` branch at commit `fce737341524d9fb884c7ee29f8ac3d8a54699a2`).

Open issues and Notes reported in the report of Gearbox V2 are not repeated in this report but may still apply. Please refer to report of the Gearbox V2 review.

2.1.1 Excluded from scope

Every contract and third-party libraries not explicitly listed above are out of scope for this review. Especially:

`core-v2`:

```
core/DataCompressor.sol
factories/*
interfaces/*
support/* except support/BlackListHelper.sol
test/*
tokens/DegenNFT.sol
```

`integrations-v2/integrations-v3`:

```
adapters/convex/ConvexV1_ClaimZap.sol
adapters/euler/*
```

```
factories/*
integrations/*
interfaces/*
multicall/*
oracles/* except oracles/curve/CurveCryptoLPPriceFeed.sol
test/*
```

The following files have been removed from the integrations codebase along with the updated version with fixes

```
adapters/aave/WrappedATokenGateway.sol
adapters/convex/ConvexV1_ClaimZap.sol
```

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 offers an updated version of its general-purposed leverage solution for ERC-20 tokens. The system is modular and consists of different parts. Gearbox V2.1 consists of the same modules as V2, the Gearbox V2 system is fully described in the Gearbox V2 report. This system overview will focus on the changes of V2.1 compared to V2.

2.2.1 The Credit System

In this update, focus has been put into reducing the attack surface by allowing users interaction with their `CreditAccount` only through the `CreditFacade`, and more specifically by forcing users to use `multicall()` for every call to either `CreditFacade` itself or any `Adapter`. Note that by using this new paradigm, users are forced to pay the gas cost for a full health check after each `multicall()`, the fast check has been deprecated. Other updates and improvements are listed below:

`CreditFacade`:

1. All the `external` functions capable of modifying the state of a user's `CreditAccount` have been removed but their functionalities are still usable through `multicall()`. One exception is for `addCollateral()`, which is still available as an `external` function.
2. A new helper contract `BlacklistHelper` has been added and is used during liquidations whenever the `CreditAccount` owner is blacklisted by `USDC` or `USDT`, which could block liquidation when the underlying is a blacklistable token. If the owner is blacklisted, the `CreditFacade` contract is given ownership of the `CreditAccount` during liquidation, so liquidation can happen, and the user can reclaim its fund to the `BlacklistHelper` later by calling `claim`. Moreover, users cannot open a `CreditAccount` if they are blacklisted.
3. Emergency liquidations are now discounted by `emergencyLiquidationDiscount`. The `emergencyLiquidationDiscount` can be set by the `CreditConfigurator`.
4. The cumulative loss of the `CreditFacade` is tracked and updated during liquidations, if needed. If the pool occurred a loss, any increase of the debt is forbidden (`isIncreaseDebtForbidden` is set to `true`), i.e., `_increaseDebt()` and opening a new `CreditAccount` are blocked. If the cumulative loss happens to go over `maxCumulativeLoss`, the `CreditManager` is paused,

meaning that the `CreditFacade` has the `isPausableAdmin` role. The `currentCumulativeLoss` can be reset by the `CreditConfigurator`. The parameters `maxCumulativeLoss` and `isIncreaseDebtForbidden` can be managed by the `CreditConfigurator`.

5. An additional sanity check is now done after a `CreditAccount` is closed. This check ensures that the pool received at least the borrowed amount plus the interest it was due. Note this does not include the fees. If it is not the case, the transaction reverts. This should already be enforced by the `CreditManager` which transfers `amountToPool` consisting of the borrowed amount, the interests and fees, but this check has been added as a second barrier.
6. The set of upgradeable contracts and related functions have been removed.

`CreditConfigurator`:

1. It is now possible to deploy a new `CreditConfigurator` for an already existing `CreditAccount`. The new `CreditConfigurator` will copy the set of allowed contracts from the old one.
2. A liquidation threshold of `0` is now allowed for the assets.
3. When allowing a new contract, if the mapping `adapterToContract[currentAdapter]` is set, the mapping will be set to `address(0)` to avoid having dangling adapters. The new function `forbidAdapter` can be used to remove dangling adapters from the already deployed system, however, this function should not be used for disabling an adapter in the system, as it will only remove one way of the two-ways mapping.
4. When upgrading the `CreditFacade`, the copy of upgradeable contracts has been removed, the total debt limit and current debt parameters, and the emergency liquidation discount are copied into the new `CreditFacade` if the `migrateParams` flag is set. If the flag is not set, the total current debt is copied and the total debt limit is set to `type(uint128).max`.
5. The function `setIncreaseForbidden` has updated access control. If `_mode` is true, the caller must be a pausable admin, if set to false, the caller must be an unpausable admin.
6. All the functions related to the upgradeble contracts have been removed.
7. New configuration functions have been added:
    - `setMaxCumulativeLoss`
    - `resetCumulativeLoss`
    - `setEmergencyLiquidationDiscount`
    - `setTotalDebtLimit`

`CreditManager`:

The code of the credit manager remains unchanged. Note that some functionality such as the fast check is no longer used. The same applies to functions featuring a paramter `convertWETH`, the updated CreditFacade now calls these functions with this parameter hardcoded to `false`.

`BoundedPriceFeed`:

This price feed reads a price from a Chainlink price oracle and upper bounds the returned price by `upperBound`. The decimals are the decimals of the queried Chainlink price oracle.

`CompositePriceFeed`:

This price feed indirectly computes the price of the target asset in USD, by using a base asset. This price feed reads two prices from Chainlink price oracle: target asset price in base denomination `T_b`, and base asset price in USD denomination `B_usd`. Then the returned price for the target asset is computed as `T_b * B_usd`. The decimals are the decimals of the `B_usd` price feed.

`CurveLPPriceFeed`:

Implements a price feed for curve crypto pools (pools of assets with volatile prices, with no expectation of price stability) with either 2 or 3 tokens. The LP price is calculated based on the prices of the pool's assets retrieved from Chainlink and the pools virtual price. Note that this pricefeed enforces that the Chainlink data is current.

2.2.2 Adapters

Adapters facilitate interaction with third party protocols using the `CreditAccounts`'s funds. All adapters have been updated to follow the new 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. Note that starting from this version, unsupported functions are no longer present. Some adapters implement variations of functions ending with `..all()`. These functions spend all balance-1 the CreditAccount has of of the spent token.

The following Adapters exist:

2.2.2.1 Yearn V2

Connects to a Yearn V2's `Vault` contract.

`YearnV2Adapter` supports the following functionality:

- `deposit()`: deposits the available balance-1 of the `Vault` underlying token into the Yearn V2 `Vault`. Enables the `yToken` and disables the underlying token.
- `deposit(amount)`: deposits the specified amount of the `Vault` underlying token into the Yearn V2 `Vault`. Enables the `yToken`.
- `deposit(amount, recipient)`: deposits the specified amount into the Yarn V2 `Vault`, the `recipient` is ignored and forced to be the `CreditAccount`. Enables the `yToken`.
- `withdraw()`: redeems the available balance-1 of `yToken` for the underlying token. Enables the underlying token and disables the `yToken`.
- `withdraw(maxShares)`: redeems the specified amount of `yToken` for the underlying token. Enables the underlying token.
- `withdraw(maxShares, recipient)`: redeems the specified amount of `yToken` for the underlying token, the `recipient` is ignored and forced to be the `CreditAccount`. Enables the underlying token.
- `withdraw(maxShares, recipient, maxLoss)`: redeems the specified amount of `yToken` for the underlying token, the `recipient` is ignored and forced to be the `CreditAccount`, the `maxLoss` parameter is forwarded to the `Vault`. Enables the underlying token.

As Yearn V2 only works with `ERC20` tokens, a `Vault` can accept deposits in `ETH`, but will never send `ETH` back.

2.2.2.2 Uniswap V2

Connects to Uniswap V2's `Router02` contract.

`UniswapV2Adapter` supports the following functionality:

- `swapTokensForExactTokens`: swaps the specified amount of `tokenOut` for at most the specified amount of `tokenIn`, following the `path`. Enables `tokenOut`.
- `swapExactTokensForTokens`: swaps the specified amount of `tokenIn` for at least the specified amount of `tokenOut`, following the `path`. Enables `tokenOut`.

- `swapAllTokensForTokens`: swaps the available balance-1 of `tokenIn` for `tokenOut` at a minimum price of `rateMinRAY`, following the `path`. Enables `tokenOut` and disables `tokenIn`.

Note that only swapping is allowed on Uniswap V2. The adapter allows up to 3 hops in the swapping path, i.e., the maximum length of the path is 4. All the intermediary tokens must be whitelisted in the adapter, this is done through `UniswapConnectorChecker`, where at most 10 tokens can be whitelisted. If not all the 10 addresses are set, `UniswapConnectorChecker` will consider `addresss(0)` as a valid hop.

2.2.2.3 Uniswap V3

Connects to Uniswap V3's `SwapRouter` contract.

`UniswapV3Adapter` supports the following functionality:

- `exactInputSingle`: swaps the specified amount of `tokenIn` for at least the specified amount of `tokenOut` on only one liquidity pool following the settings set in the provided `ExactInputSingleParams` struct. The `recipient` field of the struct is enforced to be the `CreditAccount`. Enables `tokenOut`.
- `exactAllInputSingle`: swaps the available balance-1 of `tokenIn` for `tokenOut` at a minimum price of `rateMinRAY` on only one liquidity pool following the settings set in the provided `ExactAllInputSingleParams` struct. Enables `tokenOut` and disables `tokenIn`.
- `exactInput`: swaps the specified amount of `tokenIn` for at least the specified amount of `tokenOut` on at least one liquidity pool following the settings set in the provided `ExactInputParams` struct. The `recipient` field of the struct is enforced to be the `CreditAccount`. Enables `tokenOut`.
- `exactAllInput`: swaps the available balance-1 of `tokenIn` for `tokenOut` at a minimum price of `rateMinRAY` on at least one liquidity pool following the settings set in the provided `ExactAllInputParams` struct. Enables `tokenOut` and disables `tokenIn`.
- `exactOutputSingle`: swaps the specified amount of `tokenOut` for at most the specified amount of `tokenIn` on only one liquidity pool following the settings set in the provided `ExactOutputSingleParams` struct. The `recipient` field of the struct is enforced to be the `CreditAccount`. Enables `tokenOut`.
- `exactOutput`: swaps the specified amount of `tokenOut` for at most the specified amount of `tokenIn` on at least one liquidity pool following the settings set in the provided `ExactOutputParams` struct. The `recipient` field of the struct is enforced to be the `CreditAccount`. Enables `tokenOut`.

Note that only swapping is allowed on Uniswap V3. The adapter allows up to 3 hops in the swapping path, i.e., the maximum length of the path is 4. All the intermediary tokens must be whitelisted in the adapter, this is done through `UniswapConnectorChecker`, where at most 10 tokens can be whitelisted. If not all the 10 addresses are set, `UniswapConnectorChecker` will consider `addresss(0)` as a valid hop.

2.2.2.4 Balancer V2

Connects to Balancer V2's `Vault` contract.

The Balancer system allows to use internal balances for swaps, this adapter ensures internal balances are not used. Each attached pool has a status: `NOT_ALL0WED`, `ALL0WED`, and `SWAP_ONLY`. `BalancerV2VaultAdapter` supports the following functionality:

The following actions are allowed on `ALL0WED` and `SWAP_ONLY` pools:

- `swap`: swaps `tokenIn` for `tokenOut` on only one liquidity pool, following the setting set in the `SingleSwap` struct. Enables `tokenOut`.

- `swapAll`: swaps the available balance-1 of `tokenIn` for `tokenOut` on only one liquidity pool, following the setting set in the `SingleSwapAll` struct. Enables `tokenOut` and disables `tokenIn`.
- `batchSwap`: allows one or more tokens to be traded on one or more liquidity pools, following the settings set in the `BatchSwapStep` structs, `assets` and `limits` arrays. Enables all the tokens in `assets` that have a negative `assetDelta`.

The `userData` field of the swap structs can be set freely by the `CreditAccount` owner. At the time of writing, none of the deployed pools make use of this `userData`, but it may play a role in the pools deployed in the future. Thus, Gearbox Protocol should carefully choose which pools they allow Gearbox V2.1 to interact with, since the behaviour of such pool is yet unknown when `userData` is non-zero and can cause security issues. (Documentation snapshot for Single Swap, Documentation snapshot for Batch Swaps)

The following actions are allowed on `ALL0WED` pools:

- `joinPool`: provides liquidity to a pool, following the settings set in the `JoinPoolRequest`. The `sender` and `recipient` parameters are enforced to be the `CreditAccount`. The `userData` is freely chosen by the `CreditAccount` owner. Enables the associated Balancer Pool Token.
- `joinPoolSingleAsset`: provides liquidity to a pool in a single asset. The `userData` is set to `EXACT_TOKENS_IN_FOR_BPT_OUT` for the `JoinKind`, followed by the specified amount of `tokenIn` and the minimum amount of `tokenOut`. Enables the associated Balancer Pool Token.
- `joinPoolSingleAssetAll`: provides liquidity to a pool in a single asset. The `userData` is set to `EXACT_TOKENS_IN_FOR_BPT_OUT` for the `JoinKind`, followed by the available balance-1 of `tokenIn` and the minimum amount of `tokenOut`, computed from `minRateRAY`. Enables the associated Balancer Pool Token and disables `tokenIn`.

The `userData` field encodes a member of the `JoinKind` enum, followed by specialized parameters. (Documentation snapshot)

- `exitPool`: withdraws liquidity from a pool, following the settings set in the `ExitPoolRequest`. The `sender` and `recipient` parameters are enforced to be the `CreditAccount`. The `userData` is freely chosen by the `CreditAccount` owner. Enables the pool's underlying tokens if the balance has more than 1 wei.
- `exitPoolSingleAsset`: withdraws liquidity from a pool in a single asset. The `userData` is set to `EXACT_BPT_IN_FOR_ONE_TOKEN_OUT` for the `ExitKind`, followed by the amount of `BPT` that will be burned and the index of the token to be withdrawn. Enables the pool's underlying tokens if the balance has more than 1 wei.
- `exitPoolSingleAssetAll`: withdraws liquidity from a pool in a single asset. The `userData` is set to `EXACT_BPT_IN_FOR_ONE_TOKEN_OUT` for the `ExitKind`, followed by the available balance-1 of `BPT` that will be burned and the index of the token to be withdrawn. Enables the pool's underlying tokens if the balance has more than 1 wei and disables the `BPT`.

The `userData` field encodes a member of the `ExitKind` enum, followed by specialized parameters. (Documentation snapshot)

2.2.2.5 Aave V2

`AaveV2_LendingPoolAdapter` connects to Aave V2's `LendingPool` and supports the following functionality:

- `deposit`: provides liquidity to the specified `asset` market to the extent of the specified `amount`. Enables the associated `aToken`.
- `depositAll`: provides liquidity to the specified `asset` market to the extent of the available balance-1. Enables the associated `aToken` and disables the specified `asset`.

- `withdraw`: withdraws liquidity from the specified `asset` market and burns the specified `amount` of `aToken`. If `amount==type(uint256).max`, redeems the available balance-1 of `aToken`. Enables `asset`. Disables the associated `aToken` if `amount==type(uint256).max`.
- `withdrawAll`: withdraws liquidity from the specified `asset` market and burns the available balance-1 of `aToken`. Enables `asset` and disables the associated `aToken`.

Since Gearbox V2.1 does not work with rebasing tokens as borrowable assets, Gearbox Protocol provides a wrapper for `aTokens`, `WrappedAToken`. This token wraps an `aToken` with the following exchange rate: `aT.balanceOf(this)/WaT.totalSupply`. The wrapped token is an ERC20 with additional functionality:

- `deposit`: pulls and deposits the specified amount of `aToken` in the contract and mints the corresponding amount of shares to the caller.
- `depositUnderlying`: pulls and deposits the specified amount of underlying of `aToken` in the corresponding Aave V2 market, deposits the `aToken` in the wrapper and mints the corresponding amount of shares to the caller.
- `withdraw`: burns the specified amount of shares and transfers the corresponding amount of `aToken` to the caller.
- `withdrawUnderlying`: burns the specified amount of shares, withdraws the corresponding amount of `aToken` from the Aave V2 market, and transfers the corresponding amount of underlying of `aToken` to the caller.

The `WrappedAToken` is permissionless.

Gearbox V2.1 implements a second adapter for Aave V2, that allows direct liquidity provision and `aToken` wrapping, `AaveV2_WrappedATokenAdapter`, through the `WrappedAToken` contract.

`AaveV2_WrappedATokenAdapter` connects to an `aToken` and supports the following functionality:

- `deposit`: calls `WrappedAToken.deposit()` with the specified amount of `aToken`. Enables the associated `WaToken`.
- `depositAll`: calls `WrappedAToken.deposit()` with the available balance-1 of `aToken`. Enables the associated `WaToken` and disables `aToken`.
- `depositUnderlying`: calls `WrappedAToken.depositUnderlying()` with the specified amount of `aToken` underlying token. Enables the associated `WaToken`.
- `depositUnderlyingAll`: calls `WrappedAToken.depositUnderlying()` with the available balance-1 of `aToken` underlying token. Enables the associated `WaToken` and disables the underlying token.
- `withdraw`: calls `WrappedAToken.withdraw()` with the specified amount of `WaToken`. Enables `aToken`.
- `withdrawAll`: calls `WrappedAToken.withdraw()` with the available balance-1 of `WaToken`. Enables `aToken` and disables the associated `WaToken`.
- `withdrawUnderlying`: calls `WrappedAToken.withdrawUnderlying()` with the specified amount of `WaToken`. Enables the underlying token.
- `withdrawUnderlyingAll`: calls `WrappedAToken.withdrawUnderlying()` with the available balance-1 of `WaToken`. Enables the underlying token and disables the associated `WaToken`.

Gearbox Protocol makes available a new gateway contract to provide liquidity in `aToken` on Gearbox V2.1's `WaToken` lending pools.

`WrappedATokenGateway` connects to a Gearbox's `WaToken` liquidity pool and supports the following functionality:

- `depositReferral`: pulls the specified amount of `aToken` from the caller, wraps it with `WrappedAToken.deposit()`, provides liquidity in the associated Gearbox's `WaToken` pool and transfers the corresponding amount of `dToken` to the specified `receiver`.
- `redeem`: redeems the previously approved specified amount of `dToken` from the associated liquidity pool, unwraps the `WaToken` and transfers the corresponding amount of `aToken` to the specified `receiver`.

The `WrappedATokenGateway` is permissionless.

2.2.2.6 Compound V2

`CompoundV2_CErc20Adapter` connects to a Compound V2's `cToken` that has an ERC20 token as underlying, and supports the following functionality:

- `mint`: deposits the specified amount of underlying token in the associated market. Enables `cToken`.
- `mintAll`: deposits the available balance-1 of underlying token in the associated market. Enables `cToken` and disables the underlying token.
- `redeem`: redeems the specified amount of `cToken`. Enables the underlying token.
- `redeemAll`: redeems the available balance-1 of `cToken`. Enables the underlying token and disables the `cToken`.
- `redeemUnderlying`: redeems an amount of `cToken` equals to the specified amount of underlying token, divided by the current exchange rate. Enables the underlying token.

A similar adapter is used for the ETH market, `CompoundV2_CEtherAdapter`, its target contract will be `CEtherGateway` instead of the `cETH`.

Both adapters will revert if the `cToken` returns a non-zero error code.

Since the external system expects native Ether in the ETH market, a gateway contract (`CEtherGateway`) is used as the `CreditAccount` supports wrapped ETH only. The gateway contract implements the required interfaces of the target contract, retrieves and unwraps the WETH from the `msg.sender` (the `CreditAccount`) and forwards it to the `cToken` contract. `cToken` received from Compound are transferred onwards to the `msg.sender` (the `CreditAccount`).

This gateway contract is permissionless and can be used by anyone. Moreover, it will revert if the `cToken` returns an error.

2.2.2.7 Lido

Connects to Lido.fi which enables liquidity for staked tokens.

LidoV1 Adapter:

- `submit`: stakes the given amount of WETH into Lido via the Gateway. StETH is received and enabled at the `CreditAccount`.
- `submitAll`: stakes all available WETH balance-1 of the `CreditAccount` into Lido via the Gateway. StETH is received and enabled at the `CreditAccount`. WETH is disabled as all balance is spent.

The Adapter features a limit on how much can be staked through this adapter. This limit can be changed by the configurator.

Since the external system expects native Ether a gateway contract (`LidoV1_WETHGateway`) is used as the `CreditAccount` supports wrapped ETH only. The gateway contract implements the required interfaces of the target contract, retrieves and unwraps the WETH from the `msg.sender` (the `CreditAccount`) and forwards it to the Lido contract. StETH received from Lido are transferred onwards to the `msg.sender` (the `CreditAccount`).

This gateway contract is permissionless and can be used by anyone.

WstETHV1:

- `wrap`: wraps given amount of stETH into wstETH. WstETH is enabled at the `CreditAccount`.
- `wrapAll`: wraps all available stETH balance-1 of the `CreditAccount`. WstETH is enabled at the `CreditAccount`, stETH is disabled as all balance is spent.
- `unwrap`: unwraps the given amount of wstETH into stETH. stETH is enabled at the `CreditAccount`.
- `unwrapAll`: unwraps all available stETH balance-1 of the `CreditAccount` into stETH. WsstETH is enabled at the `CreditAccount`, wstETH is disabled since all balance is spent.

WstETHGateway:

- `addLiquidity`: add stETH liquidity to the wstETH pool of Gearbox. Note that wstETH is spent from the `CreditAccount`, unwrapped and deposited into the pool. The Diesel tokens are received directly by and enabled at the `CreditAccount`.
- `removeLiquidity`: redeems LP tokens for wstETH from the pool, unwraps the wstETH into stETH which is transferred to and enabled at the `CreditAccount`.

2.2.2.8 Curve

Multiple adapters exist which facilitate the interaction with Curve.fi.

CurveV1_2 / CurveV1_3 / CurveV1_4 Adapters:

Exchanging:

- `exchange`: swaps token `i` of the curve pool (`tokenIn`) for token `j` (`tokenOut`). Enables `tokenOut`.
- `exchange_all`: swaps all balance-1 the `CreditAccount` holds of token `i` of the curve pool (`tokenIn`) for token `j` (`tokenOut`). Enables `tokenOut`, disables `tokenIn`·
- `exchange_underlying`: swaps token `i` of the curve pool (`tokenIn`) for the underlying of token `j` (`tokenOut`). Enables `tokenOut`.
- `exchange_all_underlying`: swaps all balance-1 the `CreditAccount` holds of token `i` of the curve pool (`tokenIn`) for the underlying of token `j` (`tokenOut`). Enables `tokenOut`, disables `tokenIn`·

Adding Liquidity:

- `add_liquidity`: adds liquidity based on the amounts per token specified (`tokenIn`, multiple) in exchange for lp shares (`tokenOut`). Enables `tokenOut`.
- `add_liquidity_one_coin`: add the amount of liquidity of the specified token (`tokenIn`) in exchange for lp shares. Enables `tokenOut`.
- `add_all_liquidity_one_coin`: adds all balance-1 the `CreditAccount` has of the pool token `i` specified (`tokenIn`) in exchange for lp shares `tokenOut`. Enables `tokenOut`.

Removing Liquidity

- `remove_liquidity`: redeem lp shares (`tokenIn`) for pool tokens (`tokenOut`, multiple). Enables `tokenOut`.
- `_remove_liquidity_imbalance`: redeems lp shares (`tokenIn`) for the specified pool tokens (`tokenOut`). Enables `tokenOut`.
- `remove_liquidity_one_coin`: redeems lp shares (`tokenIn`) for the specified pool token (`tokenOut`). Enables `tokenOut`.

- `remove_all_liquidity_one_coin`: redeems all lp shares (`tokenIn`) balance-1 of the `CreditAccount` for the specified pool token (`tokenOut`). Enables `tokenOut`, disables `tokenIn`.

Furthermore, these adapters provide a view function `calc_add_one_coin`. Supported Metapools must be in the form (token[0]=asset, token[1]=underlying pool).

Note that Curve.fi has some (old) pools which do not implement the default API. Whenever this adapter is deployed for a certain pool, extra care should be taken to ensure that this curve pool is compatible and implements the expected interface.

`Curve_stETH`:

This is a special version of the adapter to interact with the stETH-ETH curve pool. Since the external system works with native Ether a gateway contract (CurveV1_stETHGateway) is used as the `CreditAccount` supports wrapped ETH only. The gateway contract implements the required interfaces of the target contract, retrieves and unwraps the WETH from the `msg.sender` (the `CreditAccount`) and forwards it to the target contract. LP tokens receive are transferred onwards to the `msg.sender` (the `CreditAccount`). Ether received is wrapped and forwarded to the `msg.sender` (the `CreditAccount`).

This gateway contract is permissionless and can be used by anyone.

`CuveV1_DepositZap`:

- `remove_liquidity_one_coin`: redeems lp shares of the pool (`tokenIn`) for token `i` of the pool (`tokenOut`). Enables `tokenOut`.
- `remove_all_liquidity_one_coin`: redeems all lp shares balance-1 the `CreditAccount` holds of the pool (`tokenIn`) for token `i` of the pool (`tokenOut`). Enables `tokenOut`, disables `tokenIn`. Min amount of tokens to be received calculated based on the `rateMinRAY` passed.

Caution I: Note that the Adapter exposes further functions inherited from `CurveV1AdapterBase`, however none of these functions is present on DepositZap contracts.

Caution II: Older lending pool deposit zaps may differ in their API.

2.2.2.9 Convex

`ConvexV1_BaseRewardPool`:

Connects to a V1 `ConvexV1_BaseRewardPool`.

Since participation in a Convex reward pool isn't represented by a token, Gearbox introduces a phantom token per pool which allows the `CreditAccount` to keep track of the balance. This phantom token supports `balanceOf` only. These tokens cannot be transferred. **In case of closure/liquidation of a credit account holding such assets, these assets must be converted using the multicall functionality.**

- `stake`: stakes Convex LP tokens (`tokenIn`). The staked position is represented by the phantom token which is enabled at the `CreditAccount` (`tokenOut`).
- `stakeAll`: stakes all balance of the Convex LP token (`tokenIn`). The staked position is represented by a by the phantom token which is enabled at the `CreditAccount` (`tokenOut`). Since all balance of `tokenIn` was spent, this token is disabled.
- `getReward`: allows the `CreditAccount` to call `getReward()` on the target contract. Enables the tokens of the `rewardTokenMask`. This mask contains all reward tokens detected when the constructor ran.
- `withdraw`: unstakes and enables Convex LP tokens (`tokenOut`) at the `CreditAccount`

- `withdrawAll`: unstakes and enables Convex LP tokens (`tokenOut`) at the `CreditAccount`. Since all virtual balance of the phantom token was spent, disables the phantom token (`tokenIn`) at the `CreditAccount`.
- `withdrawAndUnwrap`: unstakes Convex LP tokens and unwraps them into Curve LP tokens. Enables Curve LP tokens (`tokenIn`) at the `CreditAccount`.
- `withdrawAllAndUnwrap`: unstakes Convex LP tokens and unwraps them into Curve LP tokens. Enables CurveLP tokens (`tokenIn`) at the `CreditAccount`. Since all virtual balance of the phantom token was spent, disables the phantom token (`tokenIn`) at the `CreditAccount`.

ConvexV1_Booster:

- `deposit`: deposits Curve LP tokens (`tokenIn`) into the booster for Convex LP tokens. Depending on the parameter `bool _stake`, stakes these Convex LP tokens into a reward pool. Hence depending on the boolean `tokenOut` is either Convex LP or the respective phantom token of the pool, this token is enabled at the `CreditAccount`.
- `depositAll`: deposits all Curve LP tokens (`tokenIn`) of the `CreditAccount` into the booster for Convex LP tokens. Depending on the parameter `bool _stake`, stakes these Convex LP tokens into a reward pool. Hence depending on the boolean, `tokenOut` is either Convex LP or the respective phantom token of the pool, this token is enabled at the `CreditAccount`.
- `withdraw`: Withdraws Curve LP (`tokenOut`) in exchange for Convex LP (`tokenIn`). Enables `tokenOut`.
- `WithdrawAll`: Withdraws Curve LP (`tokenOut`) in exchange for all Convex LP (`tokenIn`) of the `CreditAccount`. Enables `tokenOut`, disables `tokenIn`.

To track participation in a Convex reward pool a phantom token is used (see `ConvexV1_BaseRewardPool`). The `ConvexV1BoosterAdapter` keeps a mapping to track pool ids to phantom tokens. To update this mapping, the configurator can call `updateStakedPhantomTokensMap`. This iterates through all adapters of the CreditManager this ConvexV1_Booster adapter is connected to. For every `ConvexV1_BaseRewardPool` adapter found, the phantom token is querried and the mapping is updated.

2.2.3 Changes in Version 2

The `CreditFacade` has a new storage struct `totalDebt`. The current total debt is increased by `borrowedAmount` when opening a new `CreditAccount` and on `_increaseDebt()` calls, and is decreased by `borrowedAmount` when closing or liquidating a `CreditAccount`, and on `_decreaseDebt()` calls. If the current debt exceeds the `totalDebtLimit`, the current debt cannot be increased, and the transaction reverts. Note that even if the pool makes a loss during a liquidation, the current debt is reduced by `borrowedAmount`. The parameters `currentTotalDebt` and `totalDebtLimit` can be set by the `CreditConfigurator`.

2.2.4 Changes in Version 3

- The CreditManager now stores the pool it's connected to as `immutable`.
- Error `TokenIsNotInAllowedList(address)` has been changed to `TokenNotAllowedException`.
- The sanity check on the liquidity pool after a liquidation in `CreditFacade._closeLiquidateAccount()` has been simplified and no longer takes the expected liquidity into account.

2.2.5 Changes in Version 6

- For UniswapV3 adapter, pair of tokens are whitelisted along with the fee level. This means users cannot interact with UniswapV3 pools for all possible fee levels for a whitelisted pair.

2.2.6 Additions and clarifications to the Trust model

Any privileged role of the Gearbox System is fully trusted to behave honestly and correctly at all times. All supported tokens are expected to be fully vetted & audited. A broken token may break the system.

The protocol and the adapters ensure security for the Gearbox protocol. The CreditFacade offers the possibility to perform balance checks at the end of the multicalls. This is one protection for the user against unexpected behavior of the external systems the adapters used interact with (notably unexpected balance changes of tokensIn/Out). Generally, users, however, must trust the external system they interact with.

This version of Gearbox introduces support to liquidate credit accounts when the owner of the CreditAccount is blacklisted by the underlying token. While this resolves the most important issue of blocked liquidations in case of surplus underlying to be transferred to the blacklisted owner, tokens with a blacklist may lead to other problems. If system contracts such as CreditAccounts or the Pool are blacklisted, the system cannot operate as intended.

This version of Gearbox features integrations with Aave V2, whose `aTokens` are rebasing. The assumption is that the current linear interest model, the rate should only increase, but Aave governance can change that. If the rate was updated to be decreasing, this would mean a loss of value in the `CreditAccount` holding `aToken/WaToken`. This should not threaten the system as long as the decreasing rate allows the liquidators to liquidate before the system makes a loss.

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:

- 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	2

5.1 Curve Base Adapter Misconfiguration

Design Low Version 1 Risk Accepted

ISSUEIDPREFIX-001

The Curve base adapter does not sanitize `_nCoins` and could be initialized with only one coin. Such a misconfiguration would not have security implication, but the adapter is likely to revert on most of the interactions.

Risk accepted:

Gearbox Protocol states:

```
This contract is never deployed by itself, and we never have to manually enter the value
for this parameter, since it’s defined as constant in derived adapters.
```

5.2 Unusable Inherited Functions

Design Low Version 1 Risk Accepted

ISSUEIDPREFIX-002

The contract `CurveV1AdapterDeposit` inherits `CurveV1AdapterBase` but the inherited `exchange*` and functions do not exist on the Curve's deposit zappers. These functions will be available through `CurveV1AdapterDeposit` but will revert if called.

Risk accepted:

Gearbox Protocol states:

```
Potential costs of changing contracts hierarchy exceed additional deployment costs.
```

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	1

- Wrong WaToken Distribution Code Corrected

Medium-Severity Findings	2

- Compound Adapter's redeemUnderlying() Not Executed Code Corrected
- Inheriting ACLTrait Includes Pause/Unpause Code Corrected

Low-Severity Findings	5

- Inconsistent Test for Reward Token Wrapper Code Corrected
- Missing Event Code Corrected
- Query of Curve's Tricrypto Pool Virtual Price Code Corrected
- BlacklistHelper Claimable Balance Is 1 Wei off Code Corrected
- UniswapConnectorChecker Missing Sanity Check Code Corrected

6.1 Wrong `WaToken` Distribution

Correctness High Version 1 Code Corrected

ISSUEIDPREFIX-016

The exchange rate depends on the contract's balance of aTokens and the total supply of the WrappedAtokens:

```
function exchangeRate() public view override returns (uint256) {
    uint256 supply = totalSupply();
    if (supply == 0) return WAD;
    return (aToken.balanceOf(address(this)) * WAD) / supply;
}
```

In `WrappedAToken.deposit()`, the exchange rate is computed after the contract received the `aToken`, so its balance has already been updated. This leads to a wrong computation of the distributed shares or `WaToken`.

```
function deposit(uint256 assets) external override returns (uint256 shares) {
    aToken.transferFrom(msg.sender, address(this), assets);
    shares = _deposit(assets);
}

function _deposit(uint256 assets) internal returns (uint256 shares) {
```

```
    shares = (assets * WAD) / exchangeRate();
    _mint(msg.sender, shares);
    emit Deposit(msg.sender, assets, shares);
}
```

Example:

For simplicity, we assume that the exchange rate of the `aToken` is 1.

User A deposits 10 `aToken`, the computed shares are `10 / 1 = 10` since the total supply is 0. After this transaction, the contract has 10 `aToken` and the total supply is 10.

User B deposits 10 `aToken`, the computed shares are `10 / (20 / 10) = 5` because the contract already holds the new 10 `aToken`. After this transaction, the contract has 20 `aToken` and the total supply is 15.

If user A or B wants to withdraw at that point, each should get their 10 `aToken` back. But if user B withdraws, the computed amount of `aToken` he will receive is `5 * (20 / 15) = 6.666...`, which is clearly not the expected amount.

Code corrected:

The updated code does not take the balances into account anymore for the computation of the exchange rate. Now, the exchange rate is computed as the ratio of the current Aave pool's normalized income and the normalized income at `WaToken` contract deployment.

```
function exchangeRate() public view override returns (uint256) {
    return WAD * lendingPool.getReserveNormalizedIncome(address(underlying)) / _normalizedIncome;
}
```

Doing so, the contract only sees the exchange rate grow, as long as Aave's interest rate is growing, and the shares cannot be maniputaled by users of the `WaToken` contract.

6.2 Compound Adapter's `redeemUnderlying()` Not Executed

Design | Medium | Version 1 | Code Corrected

ISSUEIDPREFIX-014

In the `CompoundV2_CErc20Adapter._redeemUnderlying()` and `CompoundV2_CEtherAdapter._redeemUnderlying()` only encode the call to the target contract, but `_execute()` is not called:

```
error = abi.decode(_encodeRedeemUnderlying(amount), (uint256));
```

This has no security implications for Gearbox, but users cannot use this function.

Code corrected:

The code has been updated to execute the call:

```
error = abi.decode(_execute(_encodeRedeemUnderlying(amount)), (uint256));
```

6.3 Inheriting ACLTrait Includes Pause/Unpause

Design | Medium | Version 1 | Code Corrected

ISSUEIDPREFIX-012

The AbstractAdapter (which is inherited by all Adapters) and the BlacklistHelper inherit ACLTrait. This abstract contract implements pause functionality:

```
///@dev Pause contract
function pause() external {
    if (!_acl.isPausableAdmin(msg.sender))
        revert CallerNotPausableAdminException();
    _pause();
}

/// @dev Unpause contract
function unpause() external {
    if (!_acl.isUnpausableAdmin(msg.sender))
        revert CallerNotUnPausableAdminException();

    _unpause();
}
```

Hence contracts inheriting from ACLTrait will have external functions pause and unpause exposed. These functions may make it look like the contract can be paused - despite no function actually being pausable.

Code corrected:

The inheritance from `ACLTrait` has been removed in the `AbstractAdapter` and kept in `BlacklistHelper`. Gearbox Protocol responded:

```
Abstract adapter no longer inherits ACL trait (for adapters, it could have potentially caused problems if
we introduced some pausable functions, because credit facade is, in fact, a pausable admin, so users would
then be able to pause an adapter in the multicall; for blacklist helper there is no risk so no change)
```

6.4 Inconsistent Test for Reward Token Wrapper

Design | Low | Version 6 | Code Corrected

ISSUEIDPREFIX-013

To check whether a reward token is wrapped, a call to the `booster()` function of the contract is performed. If the call succeeds, then the reward token is further unwrapped. However, the test whether the second reward token is wrapped or not in the constructor of `ConvexV2_BaseRewardPool` is inconsistent. The check for is using `_extraReward1` instead of `_extraReward2`.

Code corrected:

Now `booster()` is called on `_extraReward2`.

6.5 Missing Event

Design Low Version 1 Code Corrected

ISSUEIDPREFIX-010

Events should be emitted whenever an important state change happens in a smart contract. Since setting `isIncreaseDebtForbidden` to `true` when the pool occurred a loss in `CreditFacade._closeLiquidatedAccount()` is an important state change, an event may be useful.

Code corrected:

If the pool occurred a loss during liquidation, the `IncurLossOnLiquidation` event is emitted.

6.6 Query of Curve's Tricrypto Pool Virtual Price

Design Low Version 1 Code Corrected

ISSUEIDPREFIX-011

In `CurveCryptoLPPriceFeed.latestRoundData()`, the virtual price is queried with `curvePool.get_virtual_price()`, but on the reference code provided by Gearbox Protocol (https://arbiscan.io/address/0x4e828A117Ddc3e4dd919b46c90D4E04678a05504#code#F3#L1) and notably in the official curve.finance pricefeed template (https://github.com/curvefi/crypto_lp_pricing/blob/b6fea6943d5ddf8648f05d442daad284c1757c86/contracts/LPPrice_tricrypto_ethereum.vy#L41), the virtual price is queried from the storage variable with `curvePool.virtual_price()`.

Code corrected:

The function `CurveCryptoLPPriceFeed.latestRoundData` has been updated to use `curvePool.virtual_price()` instead of `curvePool.get_virtual_price()`.

6.7 `BlacklistHelper` Claimable Balance Is 1 Wei off

Design Low Version 1 Code Corrected

ISSUEIDPREFIX-015

In `CreditFacade._increaseClaimableBalance()`, the parameter `balanceBefore` has 1 wei too many due to `_isBlacklisted()`. The claimable `amount` is computed as `balance-balanceBefore` and will lack 1 wei.

Code corrected:

The claimable amount has been updated to be computed as `helperBalance - helperBalanceBefore + 1;`

6.8 `UniswapConnectorChecker` Missing Sanity Check

Design | Low | Version 1 | Code Corrected

ISSUEIDPREFIX-009

The constructor of `UniswapConnectorChecker` accepts an array of addresses as parameter, but the length of the array is never checked to be `<=10`. So the checker could be deployed with an array of 25 addresses, only the 10 first will be saved in storage, but `numConnectors` will be 25. This will also incur unnecessary gas cost when `getConnectors()` is called.

Code corrected:

The constructor has been updated to revert if more than 10 addresses are provided.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Code Duplication

Informational **Version 1**

ISSUEIDPREFIX-003

The function `CurveV1StETHPoolGateway.remove_liquidity_imbalance` transfers `token0` and `token1` in the function's body, but the dedicated function `_transferAllTokensOf` can be used.

7.2 Code Inconsistencies

Informational **Version 1** **Code Partially Corrected**

ISSUEIDPREFIX-006

1. For gas optimizations, the system tries to always keep 1 wei in the balances and the standard way to check it across the codebase is with `balance <= 1`, however in `BlacklistHelper.claim()` the check is `amount < 2`.
2. The Lido gateway transfers the full balance instead of balance-1 as everywhere else in the system (gas optimization).
3. In the adapters, `_gearboxAdapterType` is sometimes overridden as a constant, and some other times as a function. For consistency across the codebase, one of the two solutions should be chosen.

Code partially corrected:

1. Changed to `amount < 1`.
2. Not addressed.
3. Not addressed.

7.3 Gas Optimizations

Informational **Version 1** **Code Partially Corrected**

ISSUEIDPREFIX-007

1. In `UniswapV2Adapter._parseUniV2Path()`, `path.length` could be loaded from memory to a local variable at the beginning of the function and read from the local variable to save a `MLOAD`.
2. In `UniswapV2Adapter._parseUniV2Path()`, the function could return early if `path.length < 2`, `path.length > 4`, or if one of the hops is not an allowed connector to save some gas.

3. In the `CurveV1AdapterBase`, the functions `add/remove_liquidity_one_coin(uint256,uint256,uint256)` do not need the `creditFacadeOnly()` modifier, since `add/remove_liquidity_one_coin(uint256,int128,uint256)` have it already.

Code partially corrected:

1. The length of the array is loaded only once at the beginning of the function and stored in a local variable.
2. The conditionnal structure has been optimized. However, the function could return early if `len > 4` to save some gas in the case of a failure.
3. The concerned functions have been updated to call the internal `_add/remove_liquidity_one_coin(int128)` which do not have the `creditFacadeOnly()` modifier.

7.4 Unused Constants

Informational **Version 1**

ISSUEIDPREFIX-004

Some of the defined constants are still declared and imported, but never used. A non-exhaustive list is:

- ALL0WANCE_THRESHOLD
- EXACT_INPUT
- EXACT_OUTPUT

7.5 Wrong Comments

Informational **Version 1** **Specification Partially Changed**

ISSUEIDPREFIX-008

Some comments in the code are wrong, here is a non-exhaustive list:

1. `WstETHGateway`: the `@notice` comment is wrong, the contract does not allow to convert `stETH` into `WstETH`, it allows to provide liquidity to Gearbox's `WstETH` in the form of `sthETH`.
2. `ACLNonReentrantTrait`: the comment of the `controllerOnly()` modifier is incomplete, it only covers the case where `externalController` is false.
3. `CreditConfigurator`: in the constructor, the call to `creditManager.upgradeCreditFacade` has a comment that specifies `Connects creditFacade and priceOracle`, but only the `CreditFacade` is connected.
4. `CurveCryptoLPPriceFeed`: the `@notice` of the `latestRoundData` function is wrong, the specified formula is not the one implemented.
5. `CreditFacade`: In `_liquidateExpiredCreditAccount` the comment "Checks if the liquidsation . . ." contains a typo.
6. The natspec of `BalancerV2VaultAdapter.batchSwap()` specifies that the `assets` must be ordered. Nothing is enforcing the ordering and Balancer V2 does not need to have the `assets` ordered.

Specifications partially corrected:

1. Not addressed.
2. The comment has been updated to include the case where `externalController` is true.
3. Not addressed.
4. The formula in the specification has been updated to match the implementation.
5. The typo has been corrected.
6. The mention of the assets' ordering has been removed.

7.6 `safeApprove` Can Revert

Informational **Version 1**

ISSUEIDPREFIX-005

Theoretically, `IERC20.safeApprove()` can revert in `WstETHGateway._checkAllowance()` and `WaToken.depositUnderlying()` because the `safeApprove()` function requires either the current allowance or the value to be 0.

- In `WstETHGateway`, the allowance for the `WstETH` token is set to `type(uint256).max` at contract deployment, and is decreased each time `WstETHGateway.addLiquidity()` is called. Also, each time `WstETHGateway.addLiquidity()` is called, the allowance check is performed, so if the allowance is strictly smaller than the amount. But the maximum allowance is such a big number that this will never happen in practice.
- In `WstETHGateway.removeLiquidity()` and `WaToken.depositUnderlying()` set the allowance for Gearbox's and Aave's lending pool to the exact amount that should be pulled from the contract. The pools are trusted to pull the exact specified amount and not less to set the allowance back to 0. If one of the pool was to be updated and pulls less than the specified amount, `WstETHGateway.removeLiquidity()` and `WaToken.depositUnderlying()` would revert.

8 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.

8.1 MetaPool With Underlying

Note Version 1

Note that there could be a Curve Metapool with a Metabpoolbase which contains an asset which has an underlying. The current CuveV1_Base implementation does not support interaction using the underlying of of one of the assets in the Metapoolbase. Gearbox Protocol stated they do not aim to support this. In practice the two most relevant base pools are 3CRV and crvFRAX, which both don't have underlyings for their assets. If such a metapool was to be added, the swap into an underlying would be supported by the router.

8.2 Multicall Reverts When Temporarily Exceeding TokenLimit

Note Version 1

Adapters don't disable `tokenIn` when uncertain whether all balance was spent. Such tokens will be disabled at the end of the multicall when the full check is executed. There is a corner case where a sequence of multicalls may revert for one credit account (as the limit would be temporarily exceeded) but not for another (where the limit is not exceeded).

This may hinder the usage of predefined multicall sequences. Note that the problem can always be rectified by adding a call to `disableToken` in between.

8.3 WrappedAToken: depositUnderlying Assumption

Note Version 1

It's of uttermost importance that the expected amount of aToken is deposited into the wrapper contract when shares are minted.

As argument `assets` the user passes the amount of underlying to `depositUnderlying()`. There is an assumption that when depositing x amount of underlying into Aave, x amount of aTokens is received in exchange. This holds if Aave works correctly as specified.

```
function depositUnderlying(uint256 assets) external override returns (uint256 shares) {
    underlying.safeTransferFrom(msg.sender, address(this), assets);
    underlying.safeApprove(address(lendingPool), assets);
    lendingPool.deposit(address(underlying), assets, address(this), 0);
    shares = _deposit(assets);
}
```

However, this makes the contract vulnerable if Aave doesn't behave as expected.

Gearbox Protocol states:

```
Wrapped aTokens will probably be deployed only for known tokens like WETH or USDC,
for which said assumption can be easily validated.
```