PUBLIC

Code Assessment

of the Gearbox V3.1 Integrations Smart Contracts

July 25, 2025

Produced for

Gearbox

by

CHAINSECURITY

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 V3.1 Integrations according to Scope to support you in forming an opinion on their security risks.

This review focuses on changes of Gearbox Protocol to adapters and integrations interacting 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. Changes between v3.0 and v3.1 of the core and their interaction with systems outside of the core, including the interaction with adapters in the scope of this review, are out of scope. Security regarding all the aforementioned subjects is high.

In Version 11, we have identified some issues regarding the Upshfit integration. Under certain conditions, the delay enforced by Upshift Vault can interfere with the expected liquidation flow, potentially leading to unexpected or increased loss for the liquidity providers. Gearbox Protocol accepted the risk and stated that it will be mitigated with proper configuration of the system.

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 would provides a high level of security. The interactions between different components of the Gearbox system are complex. 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	3
• Code Corrected	2
• Risk Accepted	1
Low-Severity Findings	5
• Code Corrected	3
• Specification Changed	1
• Risk Accepted	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 V3.1 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
6	8 Jul 2024	29c967b3ca2e17a46f73b477c570589e63365034	Added serialization, zircuit adapter, safe price checks
7	19 Jul 2024	056e52102d88a4481aef56a579df13340c434b01	Added DAI-like permit in zappers and other fixes
8	29 Jul 24	e2fcd9aef437a19a37fdbd95cde2b7df16bac346	Added IVersion for zappers and fixes
9	21 Oct 2024	361fb5c04df11a42d43bd46a2befcf98da6a8ce3	Fix for Zircuit phantom token
10	3 June 2025	217c43b301f3d64744f6955cec67ab123b3bdc39	Minor refactoring of adapters
11	14 July 2025	6a3c74f7441b9882c0bda4dd27a0459ca98cedd3	ERC4626 Zappers
12	14 July 2025	0cba624ce59ebe335c7f5de31acdb1422d90ea2f	Usphift adapters (open PR)
13	25 July 2025	883aad9cffe8ea77258f806b6ebbf34a013d9348	Release 3.1

For the solidity smart contracts, the compiler version `0.8.17` was chosen. After Version 6, the compiler version was updated to `0.8.23`.

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
```

After Version 6, the scope has been updated as follows:

- Deleted:

```
adapters:
    aave:
        AaveV2_LendingPoolAdapter.sol
        AaveV2_WrappedATokenAdapter.sol
    compound:
        CompoundV2_CErc20Adapter.sol
        CompoundV2_CEtherAdapter.sol
        CompoundV2_CTokenAdapter.sol
helpers:
    aave:
        AaveV2_WrappedAToken.sol
    compound:
        CompoundV2_CEtherGateway.sol
interfaces:
    aave:
        IAaveV2_LendingPoolAdapter.sol
        IAaveV2_WrappedATokenAdapter.sol
    compound:
        ICompoundV2_CTokenAdapter.sol
zappers:
    DTokenDepositZapper.sol
    DTokenFarmingZapper.sol
    WATokenDepositZapper.sol
    WATokenFarmingZapper.sol
    traits:
        DTokenTrait.sol
        WATokenTrait.sol
```

- Added:

```
adapters:
    zircuit:
        ZircuitPoolAdapter.sol
helpers:
    PhantomERC20.sol
    zircuit:
        ZircuitPhantomToken.sol
integrations:
    zircuit:
        IZircuitPool.sol
interfaces:
    IPhantomToken.sol
    IStateSerializer.sol
    zircuit:
        IZircuitPoolAdapter.sol
```

After Version 7, the scope has been updated as follows:

- Deleted:

```
interfaces:
    IPhantomToken.sol
    IStateSerializer.sol
```

For VERSION 10, only files that have already been inclded in scope in past versions were considered. Moreover, the changes for `CurveV1_StableNG.sol` were also reviewed. However, the core functionality of this contract has not been reviewed.

For VERSION 11, the following contracts were added:

```
adapters:
    upshift:
        UpshiftVaultAdapter.sol

helpers:
    upshift:
        UpshiftVaultGateway.sol
        UpshiftVaultWithdrawalPhantomToken.sol



zappers:
    ERC4626Zapper.sol
    StakedERC4626Zapper.sol
    traits:
        ERC4626Trait.sol
        StakedERC4626Trait.sol
```

2.1.1 Excluded from scope

Any contracts not explicitly listed above are out of the scope of this review. In particular, the following adapters are out-of-scope:

```
adapters:
    camelot:
        CamelotV3Adapter.sol
    curve:
        CurveV1_StableNG.sol
    mellow:
        Mellow4626VaultAdapter.sol
        MellowVaultAdapter.sol
    pendle:
        PendleRouterAdapter.sol
    velodrome:
        VelodromeV2RouterAdapter.sol
```

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. Updates in the core protocol not covered by the core V3 audit report are out of the scope of this review and the interactions with `integrations-v3` are assumed to work as expected.

3 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.

3.1 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.

3.1.1 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.

3.2 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.

3.3 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:

- `Underlyling[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`.

3.4 Changes in Version 3

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

3.5 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.

3.6 Changes in Version 6

- Version of the contracts was bumped from `3_00` to `3_10`.
- The `adapterType` has been replaced by `contractType`.
- Instead of returning the tokens to enable/disable, adapters now return a boolean value that indicates whether safe prices should be used in the collateral check.
- A new `serialize()` function was added, it returns the configuration of the adapter.
- The tokens used in the adapters are now only validated upon adapter deployment, addition of a supported token, or tokens approval instead of validation for all the tokens involved at each interaction.
- The adapters and zappers related to Compound and Aave have been removed.
- A new adapter for Zircuit has been added.

3.7 Changes in Version 7

- The zappers can now be used with tokens having DAI-like permit function signature, i.e. an `allowed` flag giving infinite allowance instead of an amount. The following functions are now available:

```
ZapperBase.redeemWithPermitAllowed()
ERC20ZapperBase.depositWithPermitAllowed()
ERC20ZapperBase.depositWithReferralAndPermitAllowed()
```

3.8 Changes in Version 8

- The zappers implement `IVersion`.

3.9 Changes in Version 11

- Two new zapper contracts were added: ERC4626Zapper and StakedERC4626Zapper. They let users convert ERC-4626 vault positions straight into Gearbox pool deposits in a single on-chain call. The ERC4626Zapper takes vault share tokens, redeems them for their underlying assets, and immediately deposits those assets into the target pool, while the StakedERC4626Zapper first withdraws staked shares from a 1inch farming pool before redeeming and depositing. Note that zappers cannot be used for the reverse flow.
- The ERC4626Adapter now accepts an optional `_gateway` parameter. If non-zero, it acts as the target contract and all calls will be forwarded to that gateway address instead of the vault itself, enabling protocol-specific routing (e.g., for delayed-withdrawal workflows). To support this, a new `vault()` getter was added alongside the existing `targetContract()` getter.
- An adapter for Upshift Vaults was introduced. It extends the ERC4626Adapter, making use of a dedicated gateway. It exposes the standard ERC-4626 deposit and mint functions but reverts on all withdrawal and redemption operations. To request a redemption, a user must call the `requestRedeem` function. Their Upshift vault share tokens are transferred into the gateway and

replaced in the Credit Account by a withdrawal phantom token such that collateral value is preserved even while the actual assets are time-locked. Once the required time has elapsed, invoking `claim(amount)` or `withdrawPhantomToken` pulls the underlying assets back from the gateway to the CA. Note that all phantom tokens are non-transferable. Therefore, various actions on the CA might be non-applicable while holding the Upshift phantom token. In Version 11 we extend our trust model by assuming that the Upshift gateway will not get blacklisted by the Upshift admins. Moreover, we assume that the upshift vaults, being ERC4626 implementations, have mitigated known relevant vulnerabilities such as inflation attacks.

3.10 Trust Model

The following role can be identified:

- Configurator: This is the most powerful role. The configurator contract is the timelock controlled by the GovernorV3. It is fully trusted. It can configure most system parameters. This role should be controlled by the Gearbox DAO. The configurator can be updated only by the configurator itself. The initial configurator is the deployer of the contracts. The configurator is allowed to set the parameters which are not set by the controller (see next). Setting parameters wrongly can lead to loss of funds.

For adapters indicating that safe prices don't have to be used, it is assumed that the associated oracle is carefully chosen so that its price cannot be manipulated.

Gearbox Protocol is expected to keep track of any reward token added in the Convex reward pool and take appropriate measures in case such tokens can be malicious.

4 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.

5 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.

6 Open 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	1

- Risk of Failed Liquidation Due to Upshift Exit Delay Risk Accepted

Low-Severity Findings	1

- Upshift Vault Insolvency Risk Risk Accepted

6.1 Risk of Failed Liquidation Due to Upshift Exit Delay

Design Medium Version 11 Risk Accepted

CS-GEARV3INTGRTNS-011

Upshift vault shares can only be redeemed with a delay. Let's assume an account that holds upshift vault shares alongside some other volatile asset. The account can become liquidatable due to the price of the volatile asset dropping. The exit delay seems to expose the system to the following risks:

1. **Flashloan liquidation with illiquid Upshift vault shares:**

   A liquidator might try to liquidate the position via a flashloan. Since the vault imposes an exit delay, even if a liquidator wanted to redeem those shares for underlying, they cannot do so within a single liquidation transaction. Shares need to be sent to the gateway and await the epoch delay breaking the liquidation flow. Therefore, the only option for the liquidator is to swap the shares in the secondary market provided there's enough liquidity. If that's not the case, the liquidation cannot proceed as the liquidator will make a loss.

Note that the liquidation can still proceed without a flashloan since the liquidator can withdraw the shares and redeem them at a later time.

2. **Liquidation after a redeem request**

   If *requestRedeem()* is already pending, the shares (now locked in the gateway) are non-transferrable until the delay elapses and *claim()*/*withdrawPhantomToken()* can be called. This means that the liquidator cannot even acquire the phantom token from the vault in exchange for providing the collateral to reduce the debt of the account. This means that the liquidator has no incentive to intervene as they'll make a loss. Note that during this lockup, further depreciation of remaining collateral deepens the deficit, risking bad debt for the protocol.

In practice such cases have low likelihood to arise since the CMs often impose a limit of one colleteral or only allow for collaterals that are correlated to the underlying.

Risk accepted:

Gearbox Protocol accepted the risk stating:

1. In case of active vault shares (i.e., not currently withdrawn), unless the vault experiences catastrophic value loss, it's still probably lucrative for liquidators to buy upshfit shares in exchange for underlying - they would be able to unstake and redeem them for 1:1 underlying after a time. This would be equivalent to having liquidationPremium% yield over the unstaking period (which is 1 day). This requires a more sophisticated liquidator that has some funds of their own to cover the immediate liquidation, but we believe that curators can probably arrange this. Worst case scenario, the DAO can perform these kinds of liquidations from its insurance fund.
2. Regarding already pending withdrawals. We can just minimize the risk by pricing the withdrawn collateral with a discount to ensure the user will not go red due to interest, and also have a separate credit manager for the asset to avoid poisoning the credit account with other assets that can bring it underwater.

6.2 Upshift Vault Insolvency Risk

Design Low Version 11 Risk Accepted

CS-GEARV3INTGRTNS-012

Upshift Vault’s withdrawal mechanism fixes the redemption rate at the moment when the `requestRedeem(shares)` function is called. At that time, the vault calculates the share tokens' worth in underlying assets and records that amount for the pending withdrawal. If, between `requestRedeem` and a later call to `claim()`, the vault’s underlying asset amount decreases (from slashing events, market depreciation, or protocol losses), and the total assets remaining may be insufficient to honor all outstanding requests at their originally locked-in rates.

Once the Upshift vault fails to cover those fixed-price redemptions, any call to `claim()` (or `withdrawPhantomToken()`) on the gateway will revert. In this scenario, phantom tokens become unredeemable and unbacked, leaving Credit Accounts holding tokens that no longer represent any recoverable assets. Note that the quota can be reset to zero, realising the losses.

Risk accepted:

Gearbox Protocol accepted the risk stating: "Unless the vault is hacked, its price is assumed to slowly grow, so the only way an account holding it can become liquidatable is if growth rate is slower than interest accrued for some time. To prevent this from ever happening, we set the special reserve price feed which returns the vault price multiplied by some constant factor 1-eps which would guarantee that account has enough funds to wait until withdrawal matures".

7 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open 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	2

- Wrong Token Expected by Zircuit Adapter Code Corrected
- Referrals for DAI-like Token Deposits With Permit Are Not Working Code Corrected

Low-Severity Findings	4

- Redundant Events Code Corrected
- Wrong Natspec Specification Changed
- Number of Underlying Tokens in Metapools Code Corrected
- WrappedAToken Does Not Implement Its Interface Code Corrected

Informational Findings	1

- Gas Optimizations Code Corrected

7.1 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.

7.2 Front-running the Redeem

Security High Version 1 Code Corrected

CS-GEARV3INTGRTNS-006

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.

7.3 Wrong Token Expected by Zircuit Adapter

Correctness **Medium** **Version 9** **Code Corrected**

CS-GEARV3INTGRTNS-010

When using `_tryWithdrawPhantomToken()`, the `CreditFacadeV3` is specifying the phantom token, but the Zircuit adapter is expecting the underlying token. This blocks partial liquidation and direct collateral withdrawal of the phantom token.

Code corrected:

A reverse mapping from phantom token to underlying token has been added in the adapter.

This issue was reported by Gearbox Protocol and was out of the scope of this review. However, the fix was reviewed against the core on commit `a60af54be23d308b781bda784aa0c96273f6b5ef`.

7.4 Referrals for DAI-like Token Deposits With Permit Are Not Working

Correctness **Medium** **Version 7** **Code Corrected**

CS-GEARV3INTGRTNS-001

The function `ERC20ZapperBase.depositWithReferralAndPermitAllowed()` accepts a referral code, but the call to `_deposit()` set the `withReferral` flag to false, which will make the call ignore the referral code.

Code corrected:

The `withReferral` flag in `ERC20ZapperBase.depositWithReferralAndPermitAllowed()` has been set to true.

7.5 Redundant Events

Design **Low** **Version 6** **Code Corrected**

CS-GEARV3INTGRTNS-007

In the functions `ConvexV1BoosterAdapter.updateSupportedPids()` and `ZircuitPoolAdapter.updateSupportedUnderlyings()`, the events `AddSupportedPid` and `AddSupportedUnderlying` can be emitted multiple times for the same `pid` or `underlying` every time the functions are called when a new pool or underlying is supported.

Code corrected:

The events are now emitted only if the new `pid` or `underlying` is not already in the supported set.

7.6 Wrong Natspec

Correctness | Low | Version 6 | Specification Changed

CS-GEARV3INTGRTNS-009

The Natspec of the `serialize()` function in the adapters mentions the adapter type and version, but the implementation omits that data.

Specification changed:

The Natspec has been updated.

7.7 Number of Underlying Tokens in Metapools

Correctness | Low | Version 1 | Code Corrected

CS-GEARV3INTGRTNS-008

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.

7.8 `WrappedAToken` Does Not Implement Its Interface

Design | Low | Version 1 | Code Corrected

CS-GEARV3INTGRTNS-005

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.

Code corrected:

The `WrappedAToken` has been removed from the codebase.

7.9 Gas Optimizations

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

CS-GEARV3INTGRTNS-003

1. 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.

Version 6

1. In `ConvexV1_BaseRewardPool.withdrawDiffAndUnwrap()`, the returned variable `enableSafePrices` is initialized but never used.
2. The substraction in `LidoV1.submitDiff()` can be done in an `unchecked` block.
3. In `ZircuitPoolAdapter.updateSupportedUnderlyings()`, the check `_getMaskOrRevert(token)` is redundant as `token` is taken from the collateral tokens of the `CreditManager`.

Code corrected:

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

Version 6

1. The variable has been removed.
2. The substraction is done in an `unchecked` block.
3. The checks `_getMaskOrRevert()` have been removed.

8 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.

8.1 Old Floating Compiler Version

Informational **Version 6** **Acknowledged**

CS-GEARV3INTGRTNS-002

Some files still have an old floating compiler version `^0.8.10` or `^0.8.17`. Here is a non-exhaustive list:

```
IStateSerializer.sol
CurveV1_stETHGateway.sol
integrations/*.sol
```

Acknowledged:

Gearbox Protocol answered:

```
Gateways are already deployed and work fine, so we just don't update the code (that's also why they have old pragmas etc).
```

9 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.

9.1 New Constraints on Uniswap Paths

Note Version 6

Before Version 6, intermediate tokens in the Uniswap paths only needed to be validated in the adapter, they didn't have to be collateral tokens in the CreditManager. Starting Version 6, intermediate tokens must also be collateral tokens in the CreditManager.

9.2 Partial Liquidations With Upshift Shares

Note Version 1

If an account becomes liquidatable, any liquidator can modify the holding of the account to make it healthy. During this process, the liquidator can make requests on behalf of the credit account. Such a request could be redemeption of the upshift shares. In such a case the owner of the CA will be forced to wait for until they can get access to these assets even if the redemption action didn't actually contribute to the liquidation of the account.

9.3 Phantom Token Withdrawal Does Not Claim Rewards

Note Version 6

Users must be aware that when `withdrawPhantomToken()` is called on the `ConvexV1BaseRewardPoolAdapter`, the rewards are not claimed. Rewards can be claimed with a call to `getReward()` or any other withdraw function with the `claim` boolean flag set to `true`.

9.4 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.