

Produced for



by



Contents

1	Executive Summary	3
2	Assessment Overview	5
3	System Overview	7
4	Limitations and use of report	18
5	Terminology	19
6	Findings	20
7	Resolved Findings	22
8	Informational	31
9	Notes	34

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 Core according to [Scope](#) to support you in forming an opinion on their security risks.

Gearbox Protocol implements the third version of the core Gearbox protocol, a protocol that allows users to open leveraged positions on various protocols.

The codebase has undergone a relatively large number of review iterations. These iterations included 3 brainstorming sessions with the Gearbox team where different attack vector scenarios were discussed. While our rigorous iterative process reflects our commitment to enhancing the security of the protocol, it also highlights its complexity and the need for continuous vigilance. Our client's codebase is fundamentally secure, yet our thorough approach underlines the evolving nature of security threats and our proactive stance in anticipating and mitigating potential risks.

The most critical subjects covered in our audit are the correctness of the accounting of the debt, the interest and the fees, the voting, the configuration of the system, the implementation of the quotas, the liquidation mechanism, and the opportunities to execute arbitrary code. The most important issue [Too Many Bots Can Block Liquidation](#), uncovered in the first iteration of the review, could temporarily prevent the liquidation of a credit account. The issue has been fixed. During the fixes review a critical issue [Anyone Can Redistribute The Votes](#) was uncovered which completely breaks the voting mechanism used by the system. The issues have been addressed. The most recent iterations only revealed up to medium severity issues. Hence, we find the security regarding the aforementioned subjects to be high. It is important to note that the project is significantly exposed to errors or misunderstandings in the functionality of integrated third-party systems. Reviewing these external systems for correctness was out of the scope of this audit.

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, however, we need to emphasize that the code complexity is high. Moreover, the contracts in this scope have undergone many changes during the review. This in combination with the fact that the reviews are limited in time reduces our confidence in the assessment of the system's security level.

In summary, we find that the codebase could provide a high level of security should all the issues be fixed and no more issues be uncovered during the review of their fixes.

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.

-Severity Findings	1
•	1
-Severity Findings	0
-Severity Findings	4
•	4
-Severity Findings	12
•	9
•	1
•	1
•	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 Core 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	06 June 2023	527b365c97abcf94da4872c4c05a3f1e70d6b	Initial Version
2	10 July 2023	44d2c116885f0f90725d2cc66201766141a1b23b	Fixes Review
3	25 Sept 2023	f126a7d9b0106f6b5c9b78fee7145a99e3db924c	Third Version
4	18 Oct 2023	519647cc73f74db3af3730549e450e19e994d0d8	Fourth Version
5	6 Nov 2023	605e89db1ce793a9207e642c0cc9ca70bd15f350	Version with fixes
6	11 Nov 2023	ca43d1b9bf79a0c2a71ce4ad6fdcc562bb525ba4	Updated version with fixes
7	7 Dec 2023	e16559ae82f0f24c3dc29693c444f40d676ebff9	Minor fixes
8	25 Jan 2024	44b71a51e3ffb1b8672d42bc61c72b53cc7b890b	Min health factor and collateral calculation
9	14 Feb 2024	27d05440deddb1af3f0505c5fc14721d637353f0	CreditConfiguratorV3 update
10	19 Mar 2024	9db98f7bb7876e40181a7235ca3a12dcfc08852a	CreditManagerV3_USDT address fix
11	28 Mar 2024	b2628d77f17fecf71feb77ebb038d5350f26fca7	CreditFacadeV3 Fix

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

The following contracts are included in scope:

```
core:
  AccountFactoryV3.sol
  AddressProviderV3.sol
  BotListV3.sol
  PriceOracleV3.sol
```

```

    WithdrawalManagerV3.sol
credit:
    CreditAccountV3.sol
    CreditConfiguratorV3.sol
    CreditFacadeV3.sol
    CreditManagerV3.sol
    CreditManagerV3_USDT.sol
governance:
    ControllerTimelockV3.sol
    GaugeV3.sol
    GearStakingV3.sol
    PolicyManagerV3.sol
interfaces:
    IAccountFactoryV3.sol
    IAddressProviderV3.sol
    IBotListV3.sol
    IControllerTimelockV3.sol
    ICreditAccountV3.sol
    ICreditConfiguratorV3.sol
    ICreditFacadeV3.sol
    ICreditFacadeV3Multicall.sol
    ICreditManagerV3.sol
    IExceptions.sol
    IGaugeV3.sol
    IGearStakingV3.sol
    ILinearInterestRateModelV3.sol
    IPoolQuotaKeeperV3.sol
    IPoolV3.sol
    IPriceOracleV3.sol
    ITimeLock.sol
    IVotingContractV3.sol
    IWithdrawalManagerV3.sol
external:
    IUSDT.sol
libraries:
    BalancesLogic.sol
    BitMask.sol
    CollateralLogic.sol
    CreditAccountHelper.sol
    CreditLogic.sol
    QuotasLogic.sol
    USDTFees.sol
    UnsafeERC20.sol
    WithdrawalsLogic.sol
pool:
    LinearInterestRateModelV3.sol
    PoolQuotaKeeperV3.sol
    PoolV3.sol
    PoolV3_USDT.sol
traits:
    ACLNonReentrantTrait.sol
    ACLTrait.sol
    ContractsRegisterTrait.sol
    PriceFeedValidationTrait.sol
    ReentrancyGuardTrait.sol

```

```
SanityCheckTrait.sol
USDT_Transfer.sol
```

The following files have been deleted at :

```
core:
  WithdrawalManagerV3.sol
interfaces:
  IWithdrawalManagerV3.sol
libraries:
  WithdrawalsLogic.sol
```

2.1.1 Excluded from scope

All the contracts not explicitly in scope are considered out-of-scope. Moreover, all the contracts used in the implementation such as the OpenZeppelin library are considered to work as expected. The parameters of the system are assumed to be properly set. The system integrates with external protocols through adapters which are assumed to operate properly, and their implementation is not part of the current review. The compatibility with previously deployed contracts, typically core systems of Gearbox V1 and V2, is out of the scope of this review. The configuration of the system by its admins is beyond the scope of this review. Configuration includes but is not limited to the selection of tokens which will be added to the system as well as their parametrization. Economic attacks to the protocol are beyond the scope of this review. Finally, partial liquidations were only examined from a specification perspective as there's no implementation available.

3 System Overview

This system overview describes the initially received version () 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 a general-purpose leverage protocol for ERC20 tokens. The system is modular and consists of different parts.

3.1 Credit System

The credit system consists of the following four core contracts: `CreditAccount`, `CreditManager`, `CreditFacade`, `CreditConfigurator`.

3.1.1 Credit Account

A Credit Account (CA) represents a leverage position and holds all the position's balances acting essentially as a wallet. The owner's access to this wallet is restricted as it contains additional funds borrowed from the pool. Interaction with external protocols using the funds of the credit account can be executed through the respective adapters.

3.1.2 Credit Manager

Each Credit Manager (CM) is tied to an underlying token and is connected to a liquidity pool with the same underlying token. The CM holds all the computational logic regarding the increase/decrease of the debt, the payment of accrued interest and fees, the opening, closure, and liquidation of a CA, the computation of loss and profit, and the computation of real and weighted collateral values. For each CA, it tracks the following values in the `CreditAccountInfo` struct:

- `debt`: the amount of underlying borrowed from the liquidity pool
- `cumulativeIndexLastUpdate`: the value of the base interest cumulative index the last time it was updated for the CA, i.e. on opening, debt increase, or debt decrease
- `cumulativeQuotaInterest`: accrued interest collected on the quotas remaining to be paid
- `quotaFees`: sum of the fees from the increase of quota remaining to be paid
- `enabledTokensMask`: bitmask of enabled tokens on the CA, disabled tokens will be skipped when computing the collateral value of the CA
- `flags`: indicates whether the CA has pending withdrawals and whether any bot has permission to manage the CA
- `since`: timestamp of the opening of the CA
- `borrower`: holder of the CA

All the actions done by the CM are called from the Credit Facade. The functions used to manage the state of a CA are detailed in the next section.

The parameters of the CM (allowed collateral tokens and their liquidation threshold, system's fees, quoted tokens, ...) are configured by the `CreditConfigurator`.

3.1.3 Credit Facade

The Credit Facade (CF) is the only entrypoint in the system to interact with the Credit Accounts. The available functions are:

- `openCreditAccount`: checks the debit and borrowing limits, burns the `DegenNFT` if the CF is in whitelisted mode, takes a free CA from the factory and initialize it with up-to-date parameters, moves the funds from the `Pool` to the CA, updates the cumulative index of the `Pool`, runs `_multicall()` and the full collateral check afterwards, reverts if $HF < 1$. It is not allowed to increase/decrease the debt within the multicall.
- `closeCreditAccount`: only the owner of the CA can call this function. It computes the total debt, claims the mature and immature withdrawals, and optionally interacts with adapters through a `_multicall()`. It erases all the permissions in the bot list, and computes the amount that needs to be paid to the `Pool`, as well as the profit made. Pulls the underlying tokens from the owner if the CA does not have enough of it, repays the `Pool` and updates its cumulative index and expected liquidity. Resets all the quotas to zero and decreases the expected interest the `Pool` will make from the quotas. It sends the remaining tokens to the owner and returns the CA to the factory. It is important to note that it is possible to close an unhealthy CA. During the closure, both mature and immature withdrawals are claimed and sent to a specified address. These values are not accounted for in the total value of the collateral held by the CA.
- `liquidateCreditAccount`: liquidates an undercollateralized or expired CA. It can happen in two modes: emergency and normal. During the emergency mode, only a set of whitelisted addresses can liquidate a CA. The function may first update the price feeds and then check if $HF < 1$ or the CF has expired. The withdrawals are claimed or canceled based on the mode (see `WithdrawalManager` for more). A `_multicall()` is optionally run with only calls to adapters allowed. All the bot permissions are erased. The function then proceeds to compute the amount needed to be paid to the `Pool`, the profit or the loss the pool made as well as the remaining funds that should be returned to the original borrower. The rest of the funds are sent to the liquidator who earns a part of

the CA's value. The system pulls underlying tokens from the liquidator if the CA does not have enough of it, repays the Pool, updates its cumulative index and expected liquidity, and burns shares of the treasury if the Pool incurs a loss. The quotas are reset for the account and the expected profit of the Pool is decreased. The CA is returned to the factory. If the Pool incurred a loss and in case the cumulative loss of CF becomes too big, the CF pauses and essentially enters an emergency mode. If the Pool incurred a loss, the limit of all the enabled quoted tokens of the CA will be set to 1. It is important to note that the pending withdrawals are not accounted for in the HF but the (forced) canceled ones are taken into account when the total value of a CA is calculated.

- **multicall**: only the owner of the CA can call this function. It calls `_multicall()` and runs the full collateral check afterward, which reverts if $HF < 1$.
- **botMulticall**: the caller must be a whitelisted bot and have permission to manage the CA. It calls `_multicall()` and runs the full collateral check afterward.
- **claimWithdrawals**: only the owner of the CA can call this function. It claims the mature withdrawals from the `WithdrawalManager` for a CA. The funds are sent to a specified address. If this transfer fails or the token is ETH, the funds are made available to the intended receiver through an immediate withdrawal.
- **setBotPermissions**: only the owner of the CA can call. Set arbitrary permissions in the bot list for an arbitrary contract address that has not been explicitly forbidden. This address will then be allowed to manage the CA given its permissions.

3.1.3.1 Multi-call

The `_multicall()` function is the main function to manage a CA. It can execute the following bundled actions:

- revert if some balances are lower than a given target, acts as slippage protection
- update the price oracle
- add collateral to the CA
- update the quotas. If a quota is increased, a fee is charged based on the increased amount. The Pool revenue is updated accordingly.
- initiate a withdrawal. If the `WithdrawalManager` has no delay, send the tokens directly to the user. Otherwise, process mature withdrawals and schedule a new withdrawal if a free slot is found, or revert if no slot is free. A maximum of two withdrawals can exist per CA.
- increase the debt. It pulls funds from the Pool to the CA, and updates the cumulative index tracked by the CM for the CA and the cumulative index of the Pool.
- decrease the debt. It transfers funds from the CA to the Pool, and repays the total debt in the following order: quotas fees, quotas interest, base interest, debt. It updates the cumulative index tracked by the CM for the CA and updates the cumulative index of the Pool. If the debt was increased within the same multicall, the debt cannot be decreased.
- pay the management bot. Can be done only once per `_multicall()` and only by the bots.
- set custom parameters for the full collateral check, typically a higher HF limit.
- enable a token for the CA. Can only be done with non-quoted tokens.
- disable a token for the CA. Can only be done with non-quoted tokens.
- revoke adapter allowances
- use the whitelisted adapters. The CA will be set to be the active Credit Account in the CM.

After the bundle actions, the function will enforce the slippage protection, forbid any enabling of a forbidden token, and reset the active Credit Account in the CM.

3.1.3.2 Full Collateral Check

The full collateral check ensures that a CA has enough collateral value by checking that the health factor is greater or equal to 1. The health factor (HF) is defined as:

$$\frac{TWV_{CA}^{USD} * price_{USD, underlying}}{TotalDebt_{CA}}$$

with TWV being the total weighted value of the CA:

$$TWV_{CA}^{USD} = \sum_{i \in eQT} \min(quota_{CA,i}^{USD}, tokenBalance_i^{USD}) * LT_i + \sum_{j \in eUT} tokenBalance_j^{USD} * LT_j$$

where :

- eQT and eUT are the enabled quoted tokens and the enabled unquoted tokens for the CA,
- $TotalDebt_{CA} = debt + baseInterest + quotasInterest + quotasFees + interestFees$
- $quota_{CA,i}^{USD}$ is the value in USD of the quota a credit account CA holds for the quoted token i and
- $tokenBalance_i^{USD}$ is the value in USD of the balance a credit account CA holds for the token i .

: The TWV calculation is changed in Version 6. All the price conversions to USD are assumed to yield a result with 8 decimals.

3.1.4 Credit Configurator

Used by the Configurator role to configure a CM and a CF.

On the CM, the CreditConfigurator can:

- add a new collateral token, quoted or not
- set the liquidation threshold for a token
- set a ramping liquidation threshold following a linear function during the ramping period
- set a token as quoted, enabling the quota logic for that token
- set the two-ways mapping adapter <--> 3rd party contract, allowing or forbidding adapters
- set the system's fees, i.e. fee on the interest (base and quotas), normal/expired liquidation fee, normal/expired liquidation discount
- update the liquidation threshold for the underlying token
- set the price oracle
- set the Credit Facade and migrate the old parameters if necessary
- set Credit Configurator
- set the maximum number of allowed enabled tokens per CA

On the CF, the CreditConfigurator can:

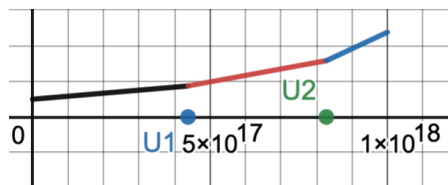
- set the allowance for a token (AllowanceAction.Allow or AllowanceAction.Forbid). A forbidden token cannot be enabled or see its balance increase during a multicall in the CF
- set the minimum and maximum debt a CA can owe the liquidity pool
- forbid further borrowing by setting the `maxDebtPerBlockMultiplier` to 0. Only the `PausableAdmin` role can do it
- set and reset the maximum cumulative loss, after which the system is paused
- set the `maxDebtPerBlockMultiplier`. Both the Controller and Configurator can do it



- set the expiration date of a CF. Both the Controller and Configurator can do it
- set the BotList address, managing the bots' permissions
- add and remove emergency liquidators, a set of whitelisted addresses that are allowed to liquidate unhealthy positions when the system is paused
- set the total debt limit, available only for old pools that do not track their debt. Both the Controller and Configurator can do it
- set the total debt parameters, available only for old pools that do not track their debt.

3.2 Pools

The pools are used to manage the liquidity of the system. Every Pool contract is tied to an underlying token, and users can lend funds to a Pool and accrue interest. The funds held by a Pool are then used by the users of the Credit Manager. Only the whitelisted Credit Managers can borrow from the Pool and send the tokens to a CA. A Pool tracks the debt of each CM that is connected to it and will revert if the debt limit is reached on a new borrow. It also tracks its current utilization through the expected and available liquidity, and the cumulative index of the base interest. The base interest rate of a Pool is based on the current utilization and follows a two-point linear function like the one drawn below:



On the liquidity providers' (LP) side, the Pool implements the ERC4626 tokenized vault standard. The LP can use the functions defined by ERC4626 to provide and remove liquidity in the Pool, the shares' computations (`totalAssets()`) are based on the expected liquidity (sum of all current LP positions + sum of unclaimed accrued base interest + sum of unclaimed quotas interest if the pool supports quotas), which is an internally tracked value that does not depend on the Pool underlying token balance. The interface of the pool includes a `depositWithReferral` function, which as the name suggests emits an event with a referral code upon deposit.

3.3 Account Factory

To reduce the costs of the deployment of the Credit Accounts, an account renting system is implemented. Upon opening a credit account, a free Credit Account contract is taken from the factory. After a position's closure, the credit account is returned to the factory. A returned credit account can only be reused after a hardcoded delay of 3 days. In case all the Credit Accounts are used, a new one is created by using the cloning paradigm.

3.4 Withdrawal Manager

The WithdrawalManager takes care of the Credit Accounts' tokens withdrawal. There are two types of withdrawals, scheduled and immediate.

- scheduled: when the owner of a CA wants to withdraw some tokens, a scheduled withdrawal is created and they must wait for a delay before the withdrawal is to be considered as mature. During the delay, the withdrawals can be processed or canceled in three ways in addition to a normal claim:
 1. On normal liquidation: the immature withdrawals are forcibly cancelled and returned to the CA and their assets are accounted for in the computation of the total value of the CA. The mature withdrawals are forcibly claimed and sent to the initial borrower. If this

transfer fails or the token is ETH, the funds are made available to the initial borrower through an immediate withdrawal.

2. On emergency liquidation: like on normal liquidation but the mature withdrawals are also cancelled.
3. On CA closure: the mature and immature withdrawals are forcibly claimed and sent to a receiver address. If this transfer fails or the token is ETH, the funds are made available for the receiver through an immediate withdrawal.

- immediate: for a given caller and token, the caller can immediately pull the funds to an arbitrary address

Each CA can have up to 2 pending scheduled withdrawals.

3.5 Quotas and Pool-Quota-Keeper

Quotas can be used if the system wants to limit its exposure to certain tokens by considering a Credit Account's balance for a quoted token only up to the quota. A maximum quoted amount can be set per token and per pool. If users want to increase the quota of a Credit Account, they must pay a fee based on the delta amount. The interest is charged at a rate decided by the Gauge. On a quota increase, the sum of the quoted amounts for a given token and for a given Pool cannot go over the maximum quoted amount for that token and that Pool. The accounting and limits for quotas are managed by the PoolQuotaKeeper (PQK) contract. There is one PQK per Pool. Quoted tokens are activated by the Configurator role.

3.6 Gauges

The Gauge contract holds the accounting of the votes of stakers that voted on the attached Pool, and computes the rates that will be applied for each quoted token, based on the votes. There is one Gauge per Pool. Only the attached voter contract (GearStakingV3) is allowed to update the votes accounting with the vote and unvote functions. When a quoted token is activated on the Gauge, one must also set $rate_{min}$ and $rate_{max}$ for that token. Stakers can either vote for $rate_{min}$ or $rate_{max}$, i.e. LP side or CA side. The PQK can be triggered to read the Gauge at the start of every new staking epoch to update the rates applied by the system to the quoted tokens of a Pool. The rate for each token is computed by the following formula:

$$rate_{min} + (rate_{max} \cdot rate_{min}) * \frac{votes_{LPside}}{votes_{LPside} + votes_{CaSide}}$$

The voter contract can be changed by the Configurator role. Quoted tokens can be activated on a Gauge by the Configurator role. The $rate_{min}$ and $rate_{max}$ for each token can be changed by Configurator role, and the effect on the system will be visible starting from the next epoch.

3.7 Staking

Users of Gearbox V3 Core can stake their GEAR tokens in GearStakingV3 and vote to agree on the interest rate that will be applied to the quoted tokens. A vote only counts once, i.e. one vote for the rate of one quoted token on one Pool. Once a withdrawal has been initiated, there is a time lock of 4 epochs before stakers can claim their GEAR tokens. The following functions are available for users:

- deposit: pull and stake GEAR tokens from the caller and optionally dispatch the voting power on one or more whitelisted Gauge contracts. The possible actions are voting and unvoting on the target Gauge contract.
- multivote: dispatch the voting power on one or more whitelisted Gauge contract. The possible actions are voting and unvoting on the target Gauge contract.

- `withdraw`: optionally dispatch the voting power and then initiate a withdrawal. The amount will be locked for the following 4 epochs before the withdrawal is mature. The withdrawn amount cannot be used to vote during the time lock period.
- `claimWithdrawals`: claim mature withdrawals and return GEAR tokens to the caller.

Each target contract can be assigned one of the following statuses: `Not_Allowed`, `Allowed` or `Unvote_Only`. The default status is `Not_Allowed`.

The `Configurator` role can update the status of the target contracts.

3.8 Bots service

It is possible for the CA owners to register with a bot service to manage their CA through the CF. The owner of a CA must first deposit some ETH in the `BotListV3` contract. Then, they can manage permissions for arbitrary, non-system-forbidden contracts, that can manage the CA, following the permissions. The bot is also allowed to take payment for its actions. The borrowers can set limits (`CreditFacade.setBotPermissions()`) for how much the bots are allowed to take as payment: an overall limit amount `fundingAmount` that is consumed until it reaches zero, and a weekly spending allowance that is automatically reset every 7 days.

Upon account closure or liquidation, all the permissions are revoked from the `BotListV3` for the involved CA. Users can withdraw the remaining funds from the contract at any time.

3.9 Timelock Controller

This contract is the entry point for most of the system setters. The general flow implemented is the following:

1. A whitelisted user submits a transaction that sets a system parameter. A transaction specifies the *target* contract, the *signature* of the method to be executed, and the parameters (*data*) to be passed as arguments.
2. The parameters are sanitized.
3. The transaction is added to the transaction queue.
4. After the required time has elapsed, and within a grace period, the submitter of the transaction can execute it.

Each transaction is associated with a policy hash. A user can submit a new transaction if they are the admins of that policy hash. A transaction can be canceled by the *vetoAdmin*.

3.10 GovernorV3

This contract is used for the management of the Gearbox protocol. Note that this contract is not part of the scope of the review but is covered by the report [Governor V3](#). We include it, however, so that the reader can get a holistic view of the system. The Governor interacts with a Timelock contract (different than the timelock contract described in the section above). The timelock contract maintains a queue of transactions which can be executed after some time. A transaction is defined by a target contract, the ETH to be attached to the call, the signature of the function to be called, the parameters of the call, and the time after which it can be executed. The Governor defines two distinct roles, the queue admin which is the DAO's multisig, and the veto admin, a multisig with a lower signature threshold than the DAO's multisig. The queue admin can:

- queue a single transaction by calling `queueTransaction()`,
- queue a batch of transactions by first calling `startBatch()` and then repeatedly `queueTransaction()`. These calls are all batched in one transaction.

The veto admin can:

- cancel a queued batch or transaction by calling `cancelBatch()` or `cancelTransaction()`.

Any user can execute a submitted transaction or batch as long as the required time has elapsed. It is important to note that a transaction cannot be executed individually if it's also part of a batch even if it was also enqueued as a single transaction.

3.11 Adapters

Adapters facilitate interaction with third-party protocols using the Credit Accounts' funds. The interactions with the adapters are done through the `CreditFacade._multicall` function. Adapters are covered in the [Gearbox V2.1](#) and in the respective report [Integrations V3](#).

3.12 Migrations of Core Components

Gearbox V3 Core consists of many non-upgradeable components-contracts. An upgrade of such a contract entails its replacement with another contract in a different address with some logic changed. Here, we present some conditions for the upgrades to be successful.

- **GearStakingV3:** requires the deployment of a new Gauge contract as well. Moreover, a migration process of the votes and the staked assets should take place.
- **GaugeV3:** all the quoted tokens from the previous gauge should be added to the new one as the pool quota keeper expects the gauge to have all quoted tokens registered. Otherwise, a new pool quota keeper must be deployed and the respective pool should also be updated.
- **Pool Quota Keeper (PQK):** the relevant pool should be updated.
- **Pool:** requires the redeployment of the connected gauge token. Users should migrate their assets themselves to a new pool. It is assumed that the credit managers associated with the pool have already returned the assets to the pool.
- **CreditFacade:** The credit facade should expire so that all the credit accounts can be liquidated. The credit manager should be updated to accept calls from the new facade.
- **CreditManager:** a redeployment of a credit facade should also take place.
- **Credit Configurator:** A new CreditConfigurator is deployed for a credit manager which copies the state of the old credit configurator. Then the configurators of the system can update the CreditConfigurator address in the credit manager.
- **WithdrawalManager:** A new credit manager should be deployed and a migration of the credit manager should take place. The assets should be claimed by their owners.
- **BotList:** the assets held by the contract should be withdrawn by their owners. The credit facade should be updated with the correct address of the new BotList contract.

3.13 Trust Model and Roles

The system defines the following roles:

- **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.

- **Controller:** This is a less powerful role that is expected to set less crucial system parameters. This role is fully trusted. It is set by the configurator. The controller is allowed to set:
 - the quota limits for the tokens
 - the fees for the quota increase
 - the total debt limit of the pool
 - the debt limit for each credit manager
 - the withdrawal fee for the liquidity providers
 - the ramp liquidation threshold
 - forbid adapters
 - the min and the max debt limit for the credit accounts
 - the max debt per block multiplier
 - the expiration date of the controller
 - the total debt limit of the facade

Should this role set some parameters wrongly, the system can get more exposed to risk than what's expected by the users. Moreover, lp's and quota users might pay very high fees.

- **(Un)PausableAdmins:** These are accounts that are allowed to pause or unpause the system. They are set by the configurator who also is the first pausable admin. The pausable admin can also forbid a token. They are fully trusted. A malicious pausable admin can temporarily block the system.
- **Emergency liquidator:** This is an address that is allowed to liquidate credit accounts when the system is paused. The configurator can set this account. As no health check is performed during emergency liquidations, these liquidators can liquidate arbitrary accounts. Emergency liquidators are fully trusted by the system.
- **Veto Admin:** They are allowed to cancel a transaction that updates some of the system parameters. The veto admin is set by the configurator. It is a fully trusted role. This admin can block the liveness of the system as far as system parameters are concerned.

The aforementioned are considered fully trusted and expected to never act maliciously.

- **End users:** Any user can be a liquidity provider or a borrower who controls a credit account. This is a non-trusted role.

Moreover, there are components of the system that are considered trusted:

- The price oracles are assumed to always return the correct prices and in the correct format. In particular, the USD prices of the tokens should always have 8 decimals. Should the price oracles return wrong prices even temporarily, credit accounts can be liquidated or withdrawals might be allowed even when those lead the credit account to an unhealthy state.
- The external contracts with which the system interacts e.g., through the adapters are considered trusted. A malicious external contract could potentially steal the assets of the users.

The contracts are not upgradeable.

Users can configure bots to execute actions on their behalf. These bots are considered trusted by the users. Moreover, the system defines some specially permissioned bots which are also fully trusted. The admin of the system can give full permission to the special bots to access funds of the credit accounts.

3.14 Changes in Version 2



- There is a maximum of 5 bots that can be approved for a `CreditAccount`.
- The function `mintWithReferral` has been added in the `PoolV3`.

3.15 Changes in Version 3

- All pools now support the quota logic.
- An account can now be opened without being forced to undertake any debt. Optionally a multicall can be executed which can add collateral to the account and increase its debt.
- An account is not allowed to have quotas if its debt at some point becomes 0. The invariant that holds now is that an account with 0 debt should continue to have 0 debt in the future until `increaseDebt()` is called again.
- An account cannot only increase or decrease its debt only once during a block. This includes decreasing the debt of an account during liquidation or closure. Such calls will fail in such cases.
- Special bots controlled by the Gearbox team are introduced. These bots can execute any call for any credit account without permissions granted.
- Gauges can be frozen and thus epochs do not progress.
- Partial liquidations: Each credit account has a health factor (HF) which should be above 1, for the account to be healthy/non-liquidatable. Ignoring quotas for a moment, the HF is the ratio of the weighted average of the balances of the enabled tokens of the system to the borrowed amount. The weight is the liquidation threshold (LT) for each token. The underlying token has the highest liquidation threshold. This means that the HF can increase should one exchange one asset with the lower LT to the underlying asset. This definition of a partial liquidation. Partial liquidations are performed by specially permissioned bots.

3.16 Changes in Version 4

- On opening of an account the debt cannot be decreased and no withdrawals can happen.
- The Withdrawal manager has been removed. Withdrawals are now executed immediately.
- The `BotList` contract has been simplified as the bots can directly get paid by withdrawing assets from the credit account.
- The security of the oracles has been further improved. When calculating the value of a collateral an option is available on whether both the main and the reserve oracle are going to be used or just the main one. In case both oracles are used the minimum price reported is taken. This safe option is used for the collateral check when collateral is withdrawn or when forbidden tokens are part of the enabled tokens. It is not used during liquidations.
- Changes in the liquidation mechanism: In the previous versions of the protocol, the liquidator had to provide the equivalent of the full value held by the CA in underlying in order to liquidate a CA.

In exchange, they would receive the assets held by the CA with some discount. As the liquidator is not expected to be that liquid, the process would be facilitated by flashloans taken at the beginning of a liquidation. At the end of the liquidation, the liquidator could exchange the assets they obtained from the CA to get back the underlying they borrowed. This could be problematic for highly illiquid tokens which are planned to be allowed by the system. In this case the liquidator couldn't swap them back to the underlying and, thus, the liquidation would be blocked as the flashloan would fail. To mitigate this risk, the liquidator must now provide only the value expected by the system after the liquidation. The liquidators get back the remaining underlying of a CA given that they're not claiming more value than they should. To claim part of the value of the assets held by the CA, the liquidators can either swap part of these assets to the underlying or immediately withdraw them. Note that, there's an implicit assumption by the system that there will always be enough liquidity even for the illiquid assets to obtain the value needed by

the system. After an account is liquidated, it is not closed i.e., returned to the factory. It's just guaranteed that its debt is 0. An account with 0 debt cannot increase its debt in the future as long as no relevant action is taken by its owner. Since accounts could now hold illiquid assets, it would be possible that liquidators are not motivated to liquidate accounts holding these assets as they're not guaranteed to make a profit.

3.17 Changes in Version 5

- On the opening of an account, withdrawals can happen.
- The minimum health factor (`fullCheckParams.minHealthFactor`) and the collateral hints checks have been moved from the `CreditManagerV3` to `CreditFacadeV3._multicall` to revert earlier if necessary.
- The function `revertIfReceivedLessThan` in the `_multicall` has been renamed to `storeExpectedBalances`.
- A new function `compareBalances` has been added in the `_multicall` to perform intermediary slippage protection if used in conjunction with the `storeExpectedBalances`.

3.18 Changes in Version 6

- The 1 wei optimization was completely removed from the `PoolQuotaKeeperV3`.
- The `maxQuotaMultiplier` was changed from 8 to 2 in `CreditFacadeV3`.
- The formula for the total weighted value (TWV) was changed to:

$$TWV_{CA}^{USD} = \sum_{i \in eQT} \min(quota_{CA,i}^{USD}, tokenBalance_i^{USD} * LT_i) + \sum_{j \in eUT} tokenBalance_j^{USD} * LT_j$$

The liquidation threshold is only applied to the token balance and not the quota.

3.19 Changes in Version 9

- The `CreditConfiguratorV3` doesn't set initial collateral tokens anymore when linked to a new `CreditManagerV3`. The collateral tokens must be added with `addCollateralToken()` afterwards.
- The check for the price oracle in `CreditConfiguratorV3._addCollateralToken()` was updated to only rely on the existence (`!= address(0)`) of the price feed in the price oracle. The validity of the price feed is done by the `PriceOracleV3` when a new price feed is added.

3.20 Changes in Version 10

- The address of the underlying token is read from the pool instead of the local `underlying`.

3.21 Changes in Version 11

- The `CreditFacadeV3._updateQuota` has been updated to round down the quote change to the closest multiple of `PERCENTAGE_FACTOR`. The change was implemented to mitigate a potential issue where a reduction in quota would not update the quota revenue expected by the pool. As a result, the pool shares would be overvalued.

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			
Medium			
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 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:

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

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

-Severity Findings	0
-Severity Findings	0
-Severity Findings	0
-Severity Findings	2

- [Calling permit on a Forbidden Token](#)
- [Missing Input Sanitization](#)

6.1 Calling `permit` on a Forbidden Token

CS-GEARV3CORE-001

Tokens can be set to forbidden on the CreditFacade. A token can be forbidden for various reasons including security issues its logic. This means the allowed interaction with it should be as constrained as possible. To that end, many actions are not permitted while a credit account holds a forbidden token such as increasing the debt of an account. Note, however, calling `addCollateralWithPermit` with 0 amount is still allowed since the credit account's balance will not be increased. During this call, a `permit()` is executed on the token. In theory, this call can execute arbitrary logic which could be dangerous for both the users and the system. Theoretically, they same concern holds when calling `transfer` and `transferFrom` on the problematic tokens.

Acknowledged:

Gearbox Protocol responded:

We do assume that tokens are not malicious because we allow users to decrease their exposure by withdrawing them or selling using adapters (both of which call `transferFrom` under the hood). And it's probably safe to assume that `permit` can't do more harm than `transferFrom`.

6.2 Missing Input Sanitization

CS-GEARV3CORE-002

1. None of the `setMaxEnabledTokens()` functions check that the new `maxEnabledTokens` is greater than zero. Setting zero as `maxEnabledTokens` would make the system unusable.

2. The `to` parameter in `GearStaking._processPendingWithdrawals` is unsanitized.
 3. The constructor of `AddressProviderV3` take the address `_acl` as parameter, but the address is not checked to be non-zero. There is no security issue as the contract would simply be unusable.
-

Code partially corrected:

1. The function `CreditConfigurator.setMaxEnabledTokens` has been updated and reverts if the new value for `maxEnabledTokens` is 0.

7 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.

-Severity Findings	1
<ul style="list-style-type: none">• Anyone Can Redistribute the Votes	
-Severity Findings	0
-Severity Findings	4
<ul style="list-style-type: none">• Custom Health Factor Is Ignored• No Rate on New Quoted Tokens• Reserve Pricefeed Can Be Main Price Feed• Too Many Bots Can Block Liquidation	
-Severity Findings	10
<ul style="list-style-type: none">• Debt Accrual on Quota Dust• Partial Liquidations When maxEnabledTokens Are Reduced• Arbitrary Bot Permissions• Debt Calculation Ignores Tokens With Fees on Transfer• Division By Zero• Global Quoted Tokens Mask Instead of Credit Account's Mask• Inconsistent Casting• Mint With Referral• Quota Increase Is Allowed for Forbidden Tokens• Updating Voter Contract May Lock GEAR	
Informational Findings	7
<ul style="list-style-type: none">• Inconsistent Remaining Balance Check• Unused Code• Wrong Comments• Interest Accrued by Quota Dust• Code Duplication• Redundant and Missing Events Emission• Inconsistent Overflow Handling	

7.1 Anyone Can Redistribute the Votes

CS-GEARV3CORE-021



The function `GearStakingV3.deposit` allows anyone to deposit GEAR tokens and execute a multivote for an arbitrary address `to`. The unrestricted call to `_multivote` in the name of `to` allows the caller to redistribute the votes of any `to` target address in the limits of `totalStaked`.

Code corrected:

The `to` parameter of the `deposit` function has been removed. The `deposit` function can only make a deposit for `msg.sender`.

7.2 Custom Health Factor Is Ignored

CS-GEARV3CORE-013

Note: This issue was discovered by Gearbox Protocol.

Users have the option to perform the collateral check with a health factor (HF) greater than 1. Let's assume that a user sets HF to 1.2. A multicall should fail if the health factor ends up being 1.1. Nevertheless, this doesn't happen as the check compares the debt to the weighted collateral ignoring the min health factor. As a result, the account is still healthy from the system's perspective.

Code corrected*

The check has been updated as follows:

```
if (cdd.twvUSD < cdd.totalDebtUSD * minHealthFactor / PERCENTAGE_FACTOR) {
    revert NotEnoughCollateralException(); // U:[CM-18B]
}
```

7.3 No Rate on New Quoted Tokens

CS-GEARV3CORE-015

When a new quoted token is added in the `PoolQuotaKeeperV3`, its rate is set to 0 by default and will keep its value until the `minRate` is set in the `GaugeV3` and a new epoch has elapsed. This would allow an attacker to request a huge quota (up to the configured limit) without paying any interest to the protocol during the period where `rate = 0`.

Code corrected:

The function `PoolQuotaKeeperV3._updateQuota` has been updated so that if `rate == 0` the `quotaChange` is zero as well, preventing users to increase their quota for a yet inactive quoted tokens.

7.4 Reserve Pricefeed Can Be Main Price Feed

CS-GEARV3CORE-024



In the function `PriceOracleV3.setReservePriceFeed()`, nothing prevents the reserve price feed to be the same as the base price feed. If the two feeds were to be the same, `LPPriceFeed.updateBounds()` could be used during a read-only reentrancy to trick the system into believing the exchange rate of the LP token is way too high.

Code corrected:

No changes related to the issue have been done in `PriceOracleV3` as it may complicate operations, but a check that the two feeds must be different has been added in `LPPriceFeed.updateBounds()`.

7.5 Too Many Bots Can Block Liquidation

CS-GEARV3CORE-027

It is possible to block a liquidation by adding enough bots so that erasing all the bot's permission on liquidation would be bigger than the block gas limit, thus blocking the liquidation process.

If such a scenario happens, a new `BotList` can always be redeployed and re-linked to the system, but all the bot users will have to withdraw their funds from the old `BotList` and set all the permissions again.

Code corrected:

A global limit of 5 bots per credit account has been added. More specifically `CreditFacade.setBotPermissions` reverts if the number of remaining bots exceeds the global limit.

7.6 Debt Accrual on Quota Dust

CS-GEARV3CORE-019

The function `QuotasLogic.calcAccruedQuotaInterest` returns 0 when `quoted <= 1` and thus does not account for the debt accrued on the quota dust. Previously, the interest accrued by 1wei left in the quota was accounted for in the pool quota revenue and materialized whenever the quota was increased again in the CA. In the updated codebase, it is still accounted for in the pool quota revenue, but will never materialize in the CA debt. This will create a small discrepancy that will increase over time between the expected pool quota revenue and the true pool quota revenue which will be lower, slightly over-evaluating the value of the LP shares.

Code corrected:

The 1wei gas optimization when a quota is set back to 0 has been removed from the codebase, fixing this issue.

7.7 Partial Liquidations When maxEnabledTokens Are Reduced

CS-GEARV3CORE-026



A (bot) multi-call cannot execute successfully if the enabled tokens at the end of the execution exceed the `maxEnabledTokens`. Consider the case where the `CreditConfigurator` sets the `maxEnabledTokens` to a value lower than the current one. This means users who have more tokens enabled should disable some of them in order to execute a (bot) multi-call. Since partial liquidations are executed by the special permissioned bot, these will be blocked as well. The specification of the partial liquidations delivered to us does not handle such a case.

Code corrected:

The codebase (`CreditManager._saveEnabledTokensMask()`) has been updated so that the underlying token is excluded from the maximum enabled tokens count, making partial liquidation by a swap to the underlying token always possible.

7.8 Arbitrary Bot Permissions

CS-GEARV3CORE-018

When setting permissions to a bot, the `permissions` are not sanitized, and it is thus possible to set arbitrary permissions that could be meaningless in the Gearbox system.

Code corrected:

A check has been added to `CreditFacadeV3.setBotPermissions` to enforce that only valid permissions can be set.

7.9 Debt Calculation Ignores Tokens With Fees on Transfer

CS-GEARV3CORE-023

Note: This issue was discovered by Gearbox Protocol.

`CreditManager.fullCollateralCheck()` calculates the total debt owed to the system denominated in the pool's underlying. If the underlying has fees on transfer, a small amount will be deducted when the debt is repaid. Even though the system is aware of this potential fee, it doesn't take it into account when it calculates the full debt during the full collateral check. This means that the health factor could be slightly overestimated allowing an account to become liquidatable slightly later than it should.

Code corrected

The fees are now accounted for in the debt calculation as follows:

```
uint256 totalDebt = _amountWithFee(cdd.calcTotalDebt());
```

7.10 Division By Zero

CS-GEARV3CORE-016

`LinearInterestRateModelV3.availableToBorrow()` calculates the `U_WAD` value by dividing with `expectedLiquidity`. However, the corner case exists for an empty pool, where the `expectedLiquidity` is 0.

Code corrected:

A check was added to take care of the case where `expectedLiquidity` is 0.

7.11 Global Quoted Tokens Mask Instead of Credit Account's Mask

CS-GEARV3CORE-014

The specification of `CreditManagerV3._getQuotedTokensData()` specifies that the returned value `_quotedTokensMask` should be the mask of the enabled quoted tokens of the Credit Account, but the actual returned value is the mask of all the quoted tokens in the Credit Manager.

Specification updated:

The nat spec has been updated to reflect the functionality of the code.

7.12 Inconsistent Casting

CS-GEARV3CORE-028

In the function `CreditConfiguratorV3._revertIfContractIncompatible()`, `_contract` is casted into `CreditFacadeV3` for the call `creditManager()`, but `_contract` can be `CreditFacadeV3`, `AdapterBase`, or `CreditConfiguratorV3` as they all implement this call.

Spec changed:

A comment has been added in `CreditConfiguratorV3._revertIfContractIncompatible()` to justify the casting clarifying that all contracts implementing this interface can be used.

7.13 Mint With Referral

CS-GEARV3CORE-020

In the `PoolV3`, the function `depositWithReferral` is available for users, but there is no such equivalent for `mint`.



Code corrected:

The function `mintWithReferral` has been added in the `PoolV3`.

7.14 Quota Increase Is Allowed for Forbidden Tokens

CS-GEARV3CORE-029

When a `CA` holds a forbidden enabled token, it is not allowed to increase its debt or the balance of such a token. However, the system allows the increase of the quota of an enabled forbidden token. While it does not increase the exposure of the system to the problematic token, it would make sense to disallow such quota updates for consistency.

Code corrected:

`CreditFacade._updateQuota` reverts when a forbidden token is specified in `callData`.

7.15 Updating Voter Contract May Lock GEAR

CS-GEARV3CORE-022

Changing the voter contract of a `Gauge` while stakers still have registered votes may break the votes accounting and prevent stakers from withdrawing their funds from the voter contract.

Code corrected:

The voter contract of a `Gauge` has been updated to be immutable.

7.16 Code Duplication

CS-GEARV3CORE-012

1. The computation of the value `pctDiff` in `PolicyManagerV3._checkPolicy()` is duplicated
 2. The computation `timestampRampStart + rampDuration` in `CreditLogic.getLiquidationThreshold()` is duplicated
-

Code corrected:

Code duplication has been removed.

7.17 Inconsistent Overflow Handling

CS-GEARV3CORE-025

`USDTFees.amountUSDTWithFee` handles the case where `maximumFee + amount` overflows. However, there is no such a case when the `amountWithBP` is calculated. This is just an inconsistency issue as it is unlikely the `amount` value to be that big.

Code corrected:

An operation which is less probable to overflow is currently used.

7.18 Inconsistent Remaining Balance Check

CS-GEARV3CORE-008

Different checks are implemented for checking if an amount is 0 or 1, `amount < 2` or `amount <= 1`. It is recommended to always use the same way of checking for consistency and code maintainability.

Code corrected:

All the checks mentioned above have been updated to be `<= 1` in the codebase.

7.19 Interest Accrued by Quota Dust

CS-GEARV3CORE-011

For gas optimization reasons, the Credit Accounts quotas are reset to 1, and this 1 wei contributes to `TokenQuotaParams.totalQuoted`. This dusty wei has multiple effects:

- this 1 wei may contribute to the pool's revenue on `updateRates` because it's based on `TokenQuotaParams.totalQuoted`, which sums up that dust.
- if a CA was closed and leaves 1 wei of quoted tokenA in `AccountQuota.quota`, the next borrower that increases the quota for that tokenA on the same CA may have some interests due because the `timeDelta * currentQuotedTokenRate` combination may be enough so that `calcAccruedQuotaInterest` may return a non zero value, recall that the minimal period before one can reuse a CA is 3 days (259200 seconds).
- if this quota is not reactivated for a long time, over multiple CA open and close cycles, the time delta between now and the last time the `accountQuota.cumulativeIndexLU` was updated can be significant.

Note, the above is a theoretical issue and mostly an inconsistency of the system and it is not expected to harm the users. It can happen that the pool revenue accounts for the 1 wei due to the reason above, but the computation in `calcAccruedQuotaInterest` yields 0.

Code corrected:



From Version 6 on, there is no 1 wei optimization so the issue is resolved.

7.20 Redundant and Missing Events Emission

CS-GEARV3CORE-017

There are multiple instances of setter functions where events are emitted regardless of whether they actually change the values. More specifically:

1. `GaugeV3.setVoter`
2. `GaugeV3.changeQuotaTokenRateParams`
3. `GearStakingV3.setVotingContractStatus`

Furthermore, an event is not emitted in `PolicyManagerV3.setPolicy`.

Code corrected:

1. The function `setVoter` and event `SetVoter` have been removed.
2. The function `GaugeV3._changeQuotaTokenRateParams` has been updated to early return and not emit any event if the updated values are the same as the old values.
3. The function `GearStakingV3.setVotingContractStatus` has been updated to early return and not emit any event if the updated value is the same as the old value.

Events have been added in `PolicyManagerV3.setPolicy()`, `PolicyManagerV3.disablePolicy()`, and `PolicyManagerV3.setGroup()`.

7.21 Unused Code

CS-GEARV3CORE-009

1. The immutable `GaugeV3.addressProvider` is set but never used.
 2. The functions `externalCall` and `approveToken` in the `CreditManagerV3` are never called.
-

Code corrected:

1. `addressProvider` has been removed.

Acknowledged:

2. Gearbox Protocol responded that these functions are not meant to be used within the current codebase, but are prep work for the future.

7.22 Wrong Comments

CS-GEARV3CORE-010

1. `USDT_Transfer._amountUSDTMinusFee()`: the value returned by the function is what the recipient would receive if amount was sent, not how much to send to reach amount
 2. `USDTFees.amountUSDTMinusFee()`: the value returned by the function is what the recipient would receive if amount was sent, not how much to send to reach amount
 3. `CreditFacadeV3._revertIfOutOfTotalDebtLimit()`:
`totalDebtLimit = totalDebt.currentTotalDebt` should be
`totalDebtLimit = totalDebt.totalDebtLimit`
 4. `CreditManagerV3.calcDebtAndCollateral()`:
Therefore, it is prevented from being called internally should be
Therefore, it is prevented from being called externally
-

Code corrected:

Comments for 1. and 2. were fixed.

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 Gas Optimizations

CS-GEARV3CORE-007

1. The computations in `GearStakingV3.getCurrentEpoch()` can be unchecked.
2. The computations of the new `withdrawalsPerEpoch` in `GearStakingV3.withdraw` can be unchecked.
3. Disabling the token outside of the `if` block in `CollateralLogic.calcNonQuotedTokensCollateral()` is redundant. There is no need to disable already disabled tokens.
4. In the function `CreditLogic.calcDecrease`, the condition `amountToRepay > quotaFees` could be transformed into a non-strict inequality to save a bit of gas since `SafeCast` will not be needed.
5. Assigning to `newDebt` and `newCumulativeIndex` in the branch `amountToRepay < cumulativeQuotaInterest + quotaProfit` of `CreditLogic.calcDecrease()` has no effect since they will be reassigned later in the function.
6. The mapping `AccountFactoryV3._queuedAccounts` could use a static array instead of a dynamic array. Its length is never read and there would not be the need for updating the length of a static array.
7. In `CollateralLogic.calcCollateral()`, one could use the first pass over the enabled token mask combined with the collateral hints on the quoted tokens, and use that information to optimize the pass on the unquoted tokens so `CollateralLogic.calcNonQuotedTokensCollateral()` does not have to iterate through the whole mask until it finds the last token.
8. The function `PoolQuotaKeeperV3._updateQuota` does not implements the gas optimization trick of leaving 1 wei when the quota is manually reduced to 0.
9. The function `ControllerTimelockV3.executeTransaction` has a branch `if signature.length==0`, but `signature` is always assigned and never empty.

Code partially corrected:

1. The computations have been moved in an unchecked block.
2. The total supply of GEAR (10_000_000_000e18) would fit in `type(uint96).max`, thus an overflow check is not needed.
3. The line `tokensToCheckMask = tokensToCheckMask.disable(tokenMask);` could be moved into the preceding `if` block to save gas. If done outside of the block, this line will be executed even for the `tokenMask` that are already disabled in the `tokensToCheckMask`.
4. The use of `SafeCast` has been removed from the `else` branch.
5. The redundant assignments have been removed.
6. The mapping has been updated to use a static array of size `2**32`.



7. This issue does not arise with a well formatted `collateralHints`, Gearbox Protocol expects well formatted `collateralHints`.
8. Gearbox Protocol specified that the optimization will be done at the UI level.

8.2 Missing Natspec

CS-GEARV3CORE-003

Some of the natspec are missing or incomplete, here is a non-exhaustive list:

1. `CreditFacadeV3.setDebtLimits()`: natspec for parameter `_maxDebtPerBlockMultiplier` is missing
2. `CreditManagerV3.closeCreditAccount()`: natspec for parameter `collateralDebtData` is missing
3. `QuotasLogic.cumulativeIndexSince()`,
`QuotasLogic.calcAdditiveCumulativeIndex()` and
`QuotasLogic.calcAccruedQuotaInterest()`: natspec for the return value are missing
4. `USDTFees`: the library is missing natspec
5. `BitMask`: the library is missing natspec for return values and the `calcIndex` function
6. `CollateralLogic.calcQuotedTokensCollateral()`: natspec for parameters `quotedTokens` and `quotasPacked` are missing

Gearbox Protocol said:

We're preparing a system-wide cleanup of NatSpec and comments.

8.3 Out-of-sync Configurators During Migration

CS-GEARV3CORE-004

When deploying a new configurator to replace an old one for a credit manager, the state of the old to-be-replaced configurator is copied i.e., the allowed adapters and the emergency liquidators. For the migration to complete another transaction to the old configurator should be executed (`CreditConfigurator.upgradeCreditConfigurator()`). In the meantime, the state of the old credit configurator could have been changed. Hence, old and new credit configurators are out of sync. The governance of the protocol should be aware of this behavior.

8.4 Pricefeed Existence Consistency

CS-GEARV3CORE-005

In `PriceOracleV3`, the checks for the existence of a price feed differ. Sometimes the check is `priceFeed != address(0)`, and sometimes `decimals != 0`.

- `getPriceRaw()`, `setReservePriceFeedStatus()` are checking for pricefeed address



- `priceFeedParams()`, `setReservePriceFeed()` are checking for decimals

8.5 Timelock Delay Can Be Zero

CS-GEARV3CORE-006

When a policy is set in `PolicyManagerV3`, nothing prevents the delay to be 0. If the delay is 0, the policy admin can set the parameter and execute it in the same transaction, which would leave no change to the veto admin to cancel the change if needed. The configurator must be careful when setting the delay for a critical policy.

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 Circular Swap With Full Balance Will Disable the Token

If a user makes a swap such $X \rightarrow Y \rightarrow X$ with the total balance of token X within the same call in the multicall, token X will be marked as disabled and unless explicitly marked as enabled with a later call in the multicall.

9.2 Fees Parameters

It is important that the fees are set correctly for the system to behave as expected. If fees are misconfigured, it could happen that certain unexpected behaviors may take place, for example, holders of unhealthy Credit Accounts could be incentivized to close their positions instead of liquidating themselves.

9.3 Quota Activation Can Make Credit Accounts Unhealthy

The activation of quota for a previously whitelisted token in the Credit Manager can make Credit Accounts using that token as active collateral liquidatable, since their quotas will be 0.

9.4 Quotas Are Independant Of Collateral

A credit account can increase its quotas by consuming all the available amount, independently of its collateral. This could make the credit account very quickly liquidatable as the quota interest will be significant. Moreover, a user can occupy the whole quota capacity and thus prevent other users from using the quota.

9.5 Quotas for Derivative Tokens

Each pool imposes a total amount of quota per token that can be used by all credit accounts related to that pool. This is done to limit the exposure of the system to specific risky assets. It's important to note that a token can derive its value from another underlying token. When setting the quota limit for either

token this dependence should be taken into account, as the exposure of the system to a risky token can end up being greater than expected.

9.6 Supported Tokens

- The system only supports standard `ERC20` tokens without special behaviors, especially tokens with callbacks (`ERC777`) which would allow arbitrary code execution. More explicitly, tokens with two entry points should also be avoided. It is important to stress, especially in the absence of a withdrawal manager, that a reentrant token could allow read-only reentrancy attacks since the state of the credit account is not properly finalized and the full collateral check hasn't been performed.
- The LP token of the `Pool` should not be allowed as collateral in the system.
- Added tokens should be reviewed regarding gas consumption. For example, the function `UnsafeERC20._unsafeCall` allows the callee to return a memory pointer that, if far in memory, would incur a huge gas cost for memory allocation.