Report

v. 3.0

Customer

Gearbox

Smart Contract Audit

Core V3

15th December 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	13.12.23	A. Zveryanskaya	Initial Draft
0.2	14.12.23	A. Zveryanskaya	Minor revision
1.0	14.12.23	A. Zveryanskaya	Release
1.1	15.12.23	A. Zveryanskaya	General recommendations minor revision
1.2	15.12.23	A. Zveryanskaya	Fix link updated
1.3	15.12.23	A. Zveryanskaya	Project scope tables merged
2.0	15.12.23	A. Zveryanskaya	Release
2.1	15.12.23	A. Zveryanskaya	Project scope table fix
3.0	15.12.23	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Gearbox Protocol brings you composable leverage. It allows anyone to margin trade on Uniswap, leverage farm on Curve, leverage liquid stake on Lido, and use 10X more capital on protocols you already love. Making decentralized leverage a reality thanks to Credit Account abstraction, bringing together lending and prime brokerage in the same protocol.

3 Project scope

We were asked to review new version of the Gearbox protocol Core V3 in the 527b365 commit:

core/			
	PriceOracleV3.sol	AccountFactoryV3.sol	AddressProviderV3.sol
	BotListV3.sol		
credit/			
	CreditAccountV3.sol	CreditConfiguratorV3.sol	CreditFacadeV3.sol
	Credit ManagerV3_USDT.sol	CreditManagerV3.sol	
governance/			
	ControllerTimelockV3.sol	GaugeV3.sol	GearStakingV3.sol
	PolicyManagerV3.sol		
interfaces/external/			
	IUSDT.sol		
interfaces/			
	IAccountFactoryV3.sol	IAddressProviderV3.sol	IBotListV3.sol
	IController TimelockV3.sol	ICreditAccountV3.sol	ICredit ConfiguratorV3.sol
	ICreditFacadeV3.sol	ICreditFacadeV3 Multicall.sol	ICreditManagerV3.sol
	IExceptions.sol	IGaugeV3.sol	IGearStakingV3.sol
	ILinearInterestRate ModelV3.sol	IPoolQuotaKeeperV3.sol	IPoolV3.sol
	IVotingContractV3.sol	IPriceOracleV3.sol	

libraries/			
	BalancesLogic.sol	BitMask.sol	CollateralLogic.sol
	CreditAccountHelper.sol	CreditLogic.sol	QuotasLogic.sol
	UnsafeERC20.sol	USDTFees.sol	
pool/			
	LinearInterestRate ModelV3.sol	PoolQuotaKeeperV3.sol	PoolV3_USDT.sol
	PoolV3.sol		
traits/			
	ACLNonReentrant Trait.sol	ACLTrait.sol	ContractsRegister Trait.sol
	ReentrancyGuard Trait.sol	SanityCheckTrait.sol	USDT_Transfer.sol
	PriceFeedValidation Trait.sol		

Then we checked protocol updates in the 527b36...f126a7d diff and the final f126a7d...605e89d diff.
General fixes were provided in the commit e16559a.

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 1 critical, 64 major, and a few less important issues. All identified Critical issues have been fixed.

Issues	Info	Fixed
Severity		
Critical	**0**	**1**
Major	**27**	**37**
Moderate	**26**	**21**
Minor	**71**	**84**

Fixed 143 out of 267 issues

6 General recommendations

Here is the list of suggestions relevant to the entire code.

1. Constant naming. We recommend following the official Solidity Style Guide and naming constants with all capital letters, using underscores to separate words.
Issue sample: PoolQuotaKeeperV3.sol

```
36 uint256 public constant override version = 3_00;
```

2. Data type. The types of state variables, function arguments, and function returns need to be specified.
Issue sample: ACLTrait.sol

```
17 address public immutable acl;
```

```
21 constructor(address addressProvider) nonZeroAddress(addressProvider)
   ↪  {
```

Issue sample: PoolV3.sol

```
157 function creditManagers() external view override returns (address[]
   ↪ memory) {
```

3. Code navigation. A few top-level constants in the project named after a library. To enhance code navigation, consider either moving the definition into the library or a separate file.
Issue sample: PoolV3.sol

```
42 struct DebtParams {
```

7 Critical Issues

CVF-1. FIXED

- **Category** Flaw
- **Source** ControllerTimelockV3.sol

Description The values “maxRateCurrent” and “minRateCurrent” were correct when transactions were enqueued and could differ from the values at the transaction execution times.

Recommendation Consider obtaining the old values at the transaction execution times.

Client Comment *Fixed by splitting the configuration function into two separate functions for each parameter.*

355
```
data: abi.encode(token, rate, maxRateCurrent)
```

379
```
data: abi.encode(token, minRateCurrent, rate)
```

8 Major Issues

CVF-2. FIXED

- **Category** Suboptimal
- **Source** PriceOracleV3.sol

Description This effectively performs multiplication after division, which could lead to precision degradation: amount * priceFrom / scaleFrom * scaleTo / priceTo

Recommendation Consider calculating as:

```
amount * priceFrom * scaleTo / (scaleFrom * priceTo)
```

or, in case scaleFrom and scaleTo are the same, as:

```
amount * priceFrom / priceTo
```

```
69 return convertFromUSD(convertToUSD(amount, tokenFrom), tokenTo); //
       ↪ U:[PO-10]
```

CVF-3. INFO

- **Category** Unclear behavior
- **Source** PriceOracleV3.sol

Recommendation It would be better to use safe conversion anyway.

Client Comment *Returning positive prices is a necessary condition for price feed to set skipCheck = true.*

```
106 // answer should not be negative (price feeds with `skipCheck = true
       ↪ ` must ensure that!)
    price = uint256(answer); // U:[PO-1]
```

CVF-4. INFO

- **Category** Unclear behavior
- **Source** SanityCheckTrait.sol

Description When a function applies this modifier to several arguments, the error gives no clue regarding which argument is zero.

Recommendation Consider passing the argument name as an argument for this modifier, and including this name into the error.

Client Comment *The current functionality of the error is sufficient for us, and expanding the scope of provided information is not worth the time required to change the error signature across the code.*

```
11 modifier nonZeroAddress(address addr) {
```

CVF-5. INFO

- **Category** Suboptimal
- **Source** ACLTrait.sol

Description This modifier is quite expensive as it performs an external call.

Recommendation Consider optimizing.

Client Comment *Retrieving the configurator dynamically allows us to seamlessly update the governance of the protocol and reduce human error. We consider the overhead an acceptable tradeoff.*

```
26 modifier configuratorOnly() {
```

CVF-6. FIXED

- **Category** Suboptimal
- **Source** USDT_Transfer.sol

Description These calls should be made only if “basisPointRate” is not zero.

Client Comment *‘amount‘ is now immediately returned if the fee is zero.*

```
26 uint256 maximumFee = IUSDT(usdt).maximumFee(); // U:[UTT_01]
   return amount.amountUSDTWithFee({basisPointsRate: basisPointsRate,
      ↪ maximumFee: maximumFee}); // U:[UTT_01]
```

```
33 uint256 maximumFee = IUSDT(usdt).maximumFee(); // U:[UTT_01]
   return amount.amountUSDTMinusFee({basisPointsRate: basisPointsRate,
      ↪ maximumFee: maximumFee}); // U:[UTT_01]
```

CVF-7. FIXED

- **Category** Procedural
- **Source** LinearInterestRateModelV3.sol

Description The comment and the variable name disagree with each other. The comment says that the number format is “WAD”, while the name ends with “_RAY”.

Recommendation Consider resolving.

Client Comment *The comment is now aligned with variable name.*

```
34 /// @notice Base interest rate in WAD format
   uint256 public immutable R_base_RAY;
```

CVF-8. FIXED

- **Category** Documentation
- **Source** PoolQuotaKeeperV3.sol

Description The comment disagree with the code.

Recommendation Consider making them consistent.

Client Comment *1 wei optimization is completely removed from PoolQuotaKeeperV3, so there's no inconsistency with the comment and function name setLimitsToZero.*

```
241 // Sets account quota to zero
```

```
243     accountQuota.quota = 1; // U: [PQK-16]
```

CVF-9. FIXED

- **Category** Suboptimal
- **Source** PoolV3.sol

Description Here, the "_amountWithFee" call is performed for the second time. The first time was inside the "withdraw" function.

Recommendation Consider refactoring to perform this call only once.

Client Comment *The value is now only computed once in the top-level external function. Also addressed for 'redeem'.*

```
350 uint256 amountToUser = _amountWithFee(assetsReceived);
```

CVF-10. INFO

- **Category** Readability
- **Source** CreditLogic.sol

Description The “amountToPool” returned value is used as a local variable, i.e. is assigned several times. This makes code harder to read.

Recommendation Consider using normal local variables for intermediate values.

Client Comment *In our opinion, directly modifying the returned value makes the process of its calculation, on the contrary, more transparent.*

```
94  amountToPool = calcTotalDebt(collateralDebtData); // U:[CL-4]
```

```
103 amountToPool += totalValue * feeLiquidation / PERCENTAGE_FACTOR;
```

```
122         amountToPool = amountMinusFeeFn(totalFunds); // U:[CL-4]
```

```
132 amountToPool = amountToPoolWithFee; // U:[CL-4]
```

CVF-11. FIXED

- **Category** Documentation
- **Source** CollateralLogic.sol

Description These arguments are not documented.

Recommendation Consider documenting them.

Client Comment *All parameters in CollateralLogic.sol are now documented.*

```
48  uint256[] memory quotasPacked,
```

```
118 address[] memory quotedTokens,
    uint256[] memory quotasPacked,
```

CVF-12. FIXED

- **Category** Documentation
- **Source** CollateralLogic.sol

Description There is no such argument.

Client Comment *This param was removed from natspec.*

```
105 // @param collateralDebtData A struct containing information on debt
       ↪  and collateral
    ///                            Quota-related data must be filled in
       ↪ for this function to work
```

CVF-13. FIXED

- **Category** Readability
- **Source** CollateralLogic.sol

Description The open brace looks like it belongs to the conditional statement, which is not actually the case.

Recommendation Consider putting blank line between the conditional statement and the open brace.

Client Comment *Blank line added.*

```
133 if (token == address(0)) break; // U:[CLL-4]
    {
```

CVF-14. FIXED

- **Category** Documentation
- **Source** CollateralLogic.sol

Description The function actually accepts two arguments.

Recommendation Consider documenting the second argument.

Client Comment *Description for the second argument added.*

```
240 /// @param collateralTokenByMaskFn A function to return collateral
       ↪ token data by its mask. Must accept inputs:
    ///                                * uint256 mask - mask of the
       ↪ token
```

CVF-15. FIXED

- **Category** Documentation
- **Source** CollateralLogic.sol

Description The comment is inconsistent with the code. The comment tells that computation is skipped for zero balance, while actually it is skipped for balance <= 1.

Recommendation Consider making the code and the comment consistent with each other.

Client Comment *See CVF-8.*

```
291 /// Collateral computations are skipped if the balance is 0
    /// and nonZeroBalance will be equal to false
    if (balance > 1) {
```

CVF-16. INFO

- **Category** Suboptimal
- **Source** CollateralLogic.sol

Description For some tokens, even one base unit could have significant value, so reducing the balance by one here could significantly affect the calculation result.

Recommendation Consider not reducing the balance, but using proper rounding instead.

Client Comment *Subtracting 1 is more correct, since on liquidation / closure most likely only 'balance - 1' of asset will be swapped. If a liquidator deliberately swaps the entire balance by passing the amount directly, the error should still be negligible - for all tokens that we encountered, a single unit is equivalent to 0.01 USD at most, and for most it is below 10^-9 USD.*

```
295 valueUSD = convertToUSDFn(priceOracle, balance - 1, token); // U:[
    ↪ CLL-1]
```

CVF-17. INFO

- **Category** Unclear behavior
- **Source** UnsafeERC20.sol

Description This reverts in case the return data is not empty but is shorter than 32 bytes.

Recommendation Consider handling this situation explicitly or at least describing it in a comment.

Client Comment *This implementation was based on* Openzeppelin. *Also, thus far we have not encountered tokens that return this kind of malformed data on transfer, so this does not appear to be a practical concern.*

```
27 return success && (returndata.length == 0 || abi.decode(returndata,
   ↪ (bool)));
```

CVF-18. INFO

- **Category** Unclear behavior
- **Source** BitMask.sol

Description This function doesn't ensure that the argument has exactly one bit set, which makes the result non-deterministic.

Recommendation Consider adding the following check: unchecked { if (mask == 0 || mask & mask - 1 != 0) revert IncorrectParameterException(); }

Client Comment *A comment was added to notify other users of the library that this is the case. Otherwise, this is an internal function and nowhere in the codebase a malformed mask is passed to it.*

```
20 function calcIndex(uint256 mask) internal pure returns (uint8 index)
   ↪  {
```

CVF-19. INFO

- **Category** Suboptimal
- **Source** BitMask.sol

Description Unrolling this loop would make the code significantly more efficient.

Recommendation Consider unrolling like this:

```
if (mask >> 128 != 0) \{mask >>= 128; index += 128;\}
if (mask >> 64 != 0) \{mask >>= 64; index += 64;\}
if (mask >> 32 != 0) \{mask >>= 32; index += 32;\}
if (mask >> 16 != 0) \{mask >>= 16; index += 16;\}
if (mask >> 8 != 0) \{mask >>= 8; index += 8;\}
if (mask >> 4 != 0) \{mask >>= 4; index += 4;\}
if (mask >> 2 != 0) \{mask >>= 2; index += 2;\}
if (mask >> 1 != 0) \{/* mask >>= 1; */ index += 1;\}
```

Client Comment *The resulting optimization does not appear to be significant and would increase contract bytesize, so the code was left as-is.*

```
27 while (true) {
```

CVF-20. FIXED

- **Category** Documentation
- **Source** BitMask.sol

Description It is unclear what happens when the same bit is set in both masks.

Recommendation Consider either forbidding such situation, or clearly defining function behavior for it.

Client Comment *Comment was added clarifying that the operations are applied consecutively.*

```
66 /// @param bitsToEnable Mask with new bits to enable
   /// @param bitsToDisable Mask with bits to disable
```

```
102 /// @param bitsToEnable Mask with new bits to enable
    /// @param bitsToDisable Mask with bits to disable
```

CVF-21. FIXED

- **Category** Suboptimal
- **Source** BalancesLogic.sol

Description This assignment is performed by reference, so all changes made in the “expected” array will also be visible in “deltas” array.

Recommendation Consider not returning anything, but just updating the passed array to make this behavior more clear.

Client Comment *We changed specification and implementation of storeBalances, and the issue is no longer present.*

```
34 expected = deltas; // U:[BLL-1]
```

CVF-22. FIXED

- **Category** Suboptimal
- **Source** BalancesLogic.sol

Recommendation There is a more efficient way to iterate through one bits in a word: while (forbiddenTokensOnAccount != 0) { uint256 tokenMask = forbiddenTokensOnAccount & uint256(-int256(forbiddenTokensOnAccount)); forbiddenTokensOnAccount &= forbidden-TokensOnAccount - 1; /* process tokenMask */ }

Client Comment *The proposed algorithm was implemented in several instances where a bit mask is being iterated.*

```
77 for (uint256 tokenMask = 1; tokenMask <= forbiddenTokensOnAccount;
   ↪ tokenMask <<= 1) {
    if (forbiddenTokensOnAccount & tokenMask != 0) {
```

CVF-23. FIXED

- **Category** Unclear behavior
- **Source** PolicyManagerV3.sol

Description These limits are applied regardless of change direction, which could lead to weird result. For example, maximum percentage change of 100% works very differently for decreases and increases.

Recommendation Consider using separate limits for increases and decreases. At least for percentages.

Client Comment *Directional bounds for percentage changes were introduced, while absolute change limits were kept bi-directional.*

```
45 uint16 minPctChange;
   uint16 maxPctChange;
```

```
48 uint256 minChange;
   uint256 maxChange;
```

CVF-24. INFO

- **Category** Suboptimal
- **Source** PolicyManagerV3.sol

Description Using the “uint16“ type for this field means that the maximum allowed percentage change is 655.35%, which could be too low in some cases.

Recommendation Consider using wider type.

Client Comment *655% is sufficient for any practical purpose.*

```
46 uint16 maxPctChange;
```

CVF-25. FIXED

- **Category** Procedural
- **Source** PolicyManagerV3.sol

Description The order of hashed values is different in the comment and in the code.

Recommendation Consider making the comment consistent with the code.

Client Comment *The comment was aligned with the code.*

```
71 /// @param policyHash A unique identifier for a policy
   ///                   Generally, this should be a hash of (
      ↪ PARAMETER_NAME, GROUP_NAME)
```

```
111     bytes32 policyHash = keccak256(abi.encode(_group[contractAddress
           ↪ ], paramName));
```

CVF-26. FIXED

- **Category** Suboptimal
- **Source** ControllerTimelockV3.sol

Description This should be executed only when the “checkPolicy” call below succeeds.

Client Comment *The call was moved to after _checkPolicy.*

```
72 uint256 totalBorrowed = pool.creditManagerBorrowed(address(
      ↪ creditManager));
```

CVF-27. INFO

- **Category** Suboptimal
- **Source** ControllerTimelockV3.sol

Description The actual transaction execution time could be later than “block.timestamp + delay”.

Recommendation Consider comparing with the real transaction execution time.

Client Comment *This is a simple sanity check to avoid completely nonsensical values being passed. rampStart being before transaction execution is handled in ‘CreditConfigurator.rampLiquidationThreshold‘.*

```
231 || rampStart < block.timestamp + delay
```

CVF-28. INFO

- **Category** Suboptimal
- **Source** ControllerTimelockV3.sol

Description Storing the whole function signature is redundant.

Recommendation Consider storing a 4-bytes function selector instead.

Client Comment *We believe it's worth keeping this as-is, to allow queued transactions to be read in a human-readable format from a contract.*

```
397 signature: signature,
```

CVF-29. FIXED

- **Category** Procedural
- **Source** GearStakingV3.sol

Description Here implicit underflow check is used to enforce a business-level constraint. This is a bad practice as it makes code harder to read and more error prone.

Recommendation Consider explicitly checking that there are enough tokens available.

Client Comment *Explicit checks were added.*

```
105 voteLockData[msg.sender].available -= amount;
```

CVF-30. FIXED

- **Category** Suboptimal
- **Source** GearStakingV3.sol

Description This reads the whole structure into the memory, while most of the read data will not be used unless epochNow > wd.epochLastUpdated.

Recommendation Consider reading the "epochLastUpdated field first, and read the remaining fields only when necessary.

Client Comment *Only wd.epochLastUpdated is read before the epoch check.*

```
123 WithdrawalData memory wd = withdrawalData[user];
```

CVF-31. INFO

- **Category** Suboptimal
- **Source** GearStakingV3.sol

Description Performing these checks on every loop iteration is suboptimal.

Recommendation Consider splitting the loop into three subsequent loops:

```
1. A loop from 0 to min(epochDiff, EPOCHS\_TO\_WITHDRAW) - 1 to
   ↪ calculate totalClaimable
2. A loop from epochDiff to EPOCHS\_TO\_WITHDRAW - 1 to copy the non
   ↪ -claimable withdrawals
3. A loop from EPOCHS\_TO\_WITHDRAW - epochDiff to EPOCHS\_TO\
   ↪ _WITHDRAW - 1 to clear tail slots
```

Client Comment *In our opinion, the proposed changed will not significantly increase either gas efficiency or readability.*

```
135 if (i < epochDiff) {
```

```
140     (i + epochDiff < EPOCHS_TO_WITHDRAW) ? wd.withdrawalsPerEpoch[i
           ↪ + epochDiff] : 0;
```

CVF-32. INFO

- **Category** Suboptimal
- **Source** GearStakingV3.sol

Recommendation Shifting non-claimable withdrawals wouldn't be necessary if the "withdrawalsPerEpoch array would be used as a ring buffer.

Client Comment *It's unclear what is meant here, as Solidity does not natively support ring buffers.*

```
139 wd.withdrawalsPerEpoch[i] =
140     (i + epochDiff < EPOCHS_TO_WITHDRAW) ? wd.withdrawalsPerEpoch[i
           ↪ + epochDiff] : 0;
```

CVF-33. FIXED

- **Category** Procedural
- **Source** GearStakingV3.sol

Description Here implicit underflow check is used to enforce a business-level constraint. This is a bad practice as it makes code harder to read and more error prone.

Recommendation Consider explicitly checking that there are enough tokens available.

Client Comment *Explicit checks were added.*

```
171 vld.available -= currentVote.voteAmount;
```

CVF-34. INFO

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Description As “degenNFT” variable is immutable, this check always returns the same result.

Recommendation Consider implementing two version of the contract, one with Degen NFT support and another without such support.

Client Comment *Modifying this would require sweeping changes across the codebase to avoid code duplication and properly abstract relevant functions. At the same time, we do not see any tangible benefit to this change, other than satisfying a preference for a particular code pattern. Therefore, we do not believe that this change is justified.*

```
245 if (degenNFT != address(0)) {
```

CVF-35. FIXED

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Recommendation As the number of applied on demand price updates is already known, it would be more efficient to remove them from “calls” before calling “_multicall”.

Client Comment *Price updates are now always performed before multicalls, and all price updates are expected to be at the beginning of ‘calls‘ array. In the main multicall body the price update calls are skipped.*

```
457 CLOSE_CREDIT_ACCOUNT_FLAGS | PRICE_UPDATES_ALREADY_APPLIED
```

CVF-36. FIXED

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Description It is unobvious here that part of the funds is actually sent not to the address specified by the “to” argument, but to the owner of the credit account.

Recommendation Consider moving the transfer of remaining funds from the “_closeCreditAccont” function to this function.

Client Comment *We changed the liquidation mechanism so that it no longer sends anything to the account owner, removing mentioned confusion.*

```
471 to: to,
```

CVF-37. INFO

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Description Comparing a value to a set of constants sequentially is suboptimal.

Recommendation Consider implementing a binary decision tree like this: if (method < boundary2) { if (method < boundary1) { ...} else {...} } else { if (method < boundary3) { ... } else { ... } }

Client Comment *Optimization in the number of comparisons does not justify decreased readability and adding new functions in the future being more complicated.*

```
633 if (method == ICreditFacadeV3Multicall.revertIfReceivedLessThan.
    ↪ selector) {

652 else if (method == ICreditFacadeV3Multicall.onDemandPriceUpdate.
    ↪ selector) {

661 else if (method == ICreditFacadeV3Multicall.addCollateral.selector)
    ↪ {

675 else if (method == ICreditFacadeV3Multicall.updateQuota.selector) {

688 else if (method == ICreditFacadeV3Multicall.scheduleWithdrawal.
    ↪ selector) {

703 else if (method == ICreditFacadeV3Multicall.increaseDebt.selector) {

717 else if (method == ICreditFacadeV3Multicall.decreaseDebt.selector) {

731 else if (method == ICreditFacadeV3Multicall.payBot.selector) {

744 else if (method == ICreditFacadeV3Multicall.setFullCheckParams.
    ↪ selector) {

753 else if (method == ICreditFacadeV3Multicall.enableToken.selector) {

767 else if (method == ICreditFacadeV3Multicall.disableToken.selector) {

782 else if (method == ICreditFacadeV3Multicall.revokeAdapterAllowances.
    ↪ selector) {

789 else {
```

CVF-38. FIXED

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Description The value ”creditManager.priceOracle()” is calculated on every price update within multicall.

Recommendation Consider refactoring to obtain the price oracle once and then reuse.

Client Comment *Price oracle is now cached in fullCheckParams and only retrieved on the first price update.*

```
910 address priceFeed = IPriceOracleV2(ICreditManagerV3(creditManager).
    ↪ priceOracle()).priceFeeds(token); // U:[FA-25]
```

CVF-39. FIXED

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Description Erasing all bots permissions here seems redundant, as no bots are supposed to have any permissions here.

Recommendation Consider either removing this call or explaining in a comment, why this call is necessary.

Client Comment *The check no longer exists in the latest ‘main‘ branch. The check was used to handle a rare misconfiguration case (updating to a previously used BotList), and can be safely removed as long as the case is avoided.*

```
1065 _eraseAllBotPermissions({creditAccount: creditAccount}); // U:[FA
    ↪ -41]
```

CVF-40. INFO

- **Category** Procedural
- **Source** CreditConfiguratorV3.sol

Description The constructor is overcomplicated and does much more than usually expected from a constructor. Apart from initializing this contract, it also able to migrate data from previous instance or initialize a credit manager instance.

Recommendation Consider moving data migration and credit manager initializaiton logic into separate functions that could be called after deploying the credit configurator contract.

Client Comment *This constructor was designed to reduce deployment complexity and human error. We do not see any compelling reason to give this up.*

```
96 constructor(CreditManagerV3 _creditManager, CreditFacadeV3
     ↪ _creditFacade, CreditManagerOpts memory opts)
```

CVF-41. INFO

- **Category** Unclear behavior
- **Source** CreditConfiguratorV3.sol

Description Copying data from the previous credit configurator here seems like a bad idea for a number of reasons: 1. The data copying is not synchronized with changing the credit configurator for the credit manager, so changes made in the old credit configurator after deployment of the new credit configurator, but before switching the credit manager to the new configurator, won't be copied. 2. The old credit configurator may be broken or hacked, or may have inconsistent data. Copying data from it could be impossibe or could break the new credit configurator. 3. It is always possible to perform copying off-chain, no need to do it on-chain.

Client Comment *Copying the data on-chain helps prevent human error. Updating the old configurator state after deploying the new one would be a very deliberate error on the part of system governance, so we don't believe it is justified to lose existing benefits to prevent it.*

```
109 // In the case where the CC is deployed for the existing Credit
    ↪ Manager,
110 // we only need to copy several array parameters from the last CC,
    // but the existing configs must be kept intact otherwise
    // 1. Allowed contracts set stores all the connected third-party
    ↪ contracts - currently only used
    //    to retrieve externally
    // 2. Emergency liquidator set stores all emergency liquidators -
    ↪ used for parameter migration when changing the Credit Facade
    // 3. Forbidden token set stores all forbidden tokens - used for
    ↪ parameter migration when changing the Credit Facade
```

CVF-42. INFO

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Description This check is quite expensive. It optimizes a very rare case, but make the most common case more expensive.

Recommendation Consider removing this check.

Client Comment *This check is primarily required to avoid firing events without any state change, as this may unnecessarily trigger monitoring systems. As this is a configuration function, the gas overhead is acceptable.*

582
```
if (
    (feeInterest == _feeInterestCurrent) && (feeLiquidation ==
       ↪ _feeLiquidationCurrent)
        && (liquidationDiscount == _liquidationDiscountCurrent)
        && (feeLiquidationExpired == _feeLiquidationExpiredCurrent)
        && (liquidationDiscountExpired ==
           ↪ _liquidationDiscountExpiredCurrent)
) return;
```

CVF-43. INFO

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Description The migration logic seems redundant, as one could migrate all necessary parameters off-chain. Also, there is no granularity, so parameters are migrated in all-or-one fashion, which limits flexibility of the migration logic.

Recommendation Consider removing the migration functionality.

Client Comment *This is intended to update the Credit Facade seamlessly and prevent human error during parameter migration. As all significant changes in the system are under a timelock, any single parameter mistake can be costly.*

671
```
if (migrateParams) {
```

CVF-44. FIXED

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Description Total debt should be copied only if the new credit facade does track total debt. Also, in case the new credit facade does track total debt, but the previous facade doesn't, migration shouldn't be allowed.

Client Comment *'trackTotalDebt' logic was removed, so this is no longer an issue.*

```
699 if (prevCreditFacace.trackTotalDebt()) {
```

CVF-45. INFO

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Description This function copies the forbidden tokens mask bit by bit.

Recommendation Consider copying the whole word at once.

Client Comment *We are more comfortable with the current 'setTokenAllowance' function that sets the forbidden status per token, as this prevents errors. As this is a configuration function, the overhead is acceptable.*

```
724 function _migrateForbiddenTokens(address _creditFacade, uint256
      ↪ forbiddenTokenMask) internal {
```

CVF-46. FIXED

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Description This loop should start at mask=2, as underlying doesn't need to be migrated.

Client Comment *The loop now starts at 2.*

```
726 for (uint256 mask = 1; mask <= forbiddenTokenMask; mask <<= 1) {
```

CVF-47. INFO

- **Category** Unclear behavior
- **Source** CreditConfiguratorV3.sol

Description There is no range check for the argument.

Recommendation Consider adding appropriate check.

Client Comment *The (0, uint8.max) range enforced by the input type is sufficient.*

827
```
function setMaxDebtPerBlockMultiplier(uint8
    ↪ newMaxDebtLimitPerBlockMultiplier)
```

CVF-48. INFO

- **Category** Unclear behavior
- **Source** CreditConfiguratorV3.sol

Description There is no range check for the “newMaxDebtLimitPerBlockMultiplier” argument.

Recommendation Consider adding appropriate check.

Client Comment *See above.*

836
```
function _setMaxDebtPerBlockMultiplier(address _creditFacade, uint8
    ↪ newMaxDebtLimitPerBlockMultiplier) internal {
```

CVF-49. INFO

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Description Obtaining the minDebt and maxDebt values from the credit facade just to pass them back is waste of gas.

Recommendation Consider implementing a separate function in the “ICreditFacadeV3” interface to set the “maxDebtLimitPerBlockMultiplier” value separately.

Client Comment *Credit Facade is bottlenecked on size, so we would prefer to avoid adding new functions unnecessarily. This is a configuration function, so the gas overhead is acceptable.*

842
```
(uint128 minDebt, uint128 maxDebt) = cf.debtLimits();
cf.setDebtLimits(minDebt, maxDebt, newMaxDebtLimitPerBlockMultiplier
    ↪ ); // I:[CC-24]
```

CVF-50. FIXED

- **Category** Procedural
- **Source** CreditConfiguratorV3.sol

Description This check should be performed by the credit facade as it belongs to its internal business logic.

Client Comment *'trackTotalDebt'-related logic was removed, so this is no longer an issue.*

965
```
if (!cf.trackTotalDebt()) revert TotalDebtNotTrackedException(); //
   ↪ I:[CC-34]
```

979
```
if (!cf.trackTotalDebt()) revert TotalDebtNotTrackedException(); //
   ↪ I:[CC-34]
```

CVF-51. INFO

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Description Passing an additional argument is cheap, external calls and storage access are expensive, so writing the active credit account into the storage via an external call could be more expensive then passing it with every operation as an additional argument.

Client Comment *This is primarily intended to avoid an incorrect CA being passed intentionally or unintentionally by an adapter. In principle, adapters can be developed by third parties, so they are less trusted than Credit Facade, and the less parameters they control, the better.*

126
```
/// _activeCreditAccount is used to avoid adapters having to
   ↪ manually pass the Credit Account
```

CVF-52. FIXED

- **Category** Documentation
- **Source** CreditManagerV3.sol

Description The arguments of the “takeCreditAccount” function don’t nave names, so their semantics is unclear.

Recommendation Consider giving them descriptive names and/or explaining their semantics in a comment.

Client Comment *Added a comment explaining that these are dummy inputs for backward compatibility.*

```
244 creditAccount = IAccountFactoryBase(accountFactory).
       ↪ takeCreditAccount(0, 0); // U:[CM-6]
```

CVF-53. FIXED

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Description This check doesn’t save gas as external call is expensive and setting a storage slot to the current value is cheap.

Recommendation Consider removing this check and just setting allowance to zero.

Client Comment *Redundant check was removed.*

```
589 uint256 allowance = IERC20(token).allowance(creditAccount, spender);
       ↪  // U:[CM-15]
591 if (allowance > 1) {
```

CVF-54. FIXED

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Recommendation There is an efficient algorithm to iterate through non-zero bits in a word: while (tokensToCheckMask != 0) { uint256 tokenMask; unchecked { // Find out the lowest non-zero bit tokenMask = tokensToCheckMask & uint256(-int256(tokensToCheckMask)); // Clear the lowest non-zero bit tokensToCheckMask &= tokensToCheckMask - 1; } // Process tokenMask }

Client Comment *The algorithm is implemented for tokens outside of collateralHints[i] during collateral computations.*

945
```
        tokenMask = (i < len) ? collateralHints[i] : 1 << (i - len);
```

947
```
        if (tokensToCheckMask & tokenMask != 0) {
```

1131
```
for (
    uint256 tokenMask = UNDERLYING_TOKEN_MASK; tokenMask <=
        ↪ tokensToTransferMask; tokenMask = tokenMask << 1
) {
    // enabledTokensMask & tokenMask == tokenMask when the token is
        ↪ enabled, and 0 otherwise
    if (tokensToTransferMask & tokenMask != 0) {
```

CVF-55. INFO

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Description The idea of getting the token mask for a token and then immediately getting token by the mask seems weird and suboptimal.

Recommendation Consider refactoring.

Client Comment *All token data is stored using the tokenMask as uid, and retrieving the token with the LT does not incur any extra gas cost. Storing LT separately from the token for this specific case would increase gas costs in several other places.*

1203
```
uint256 tokenMask = getTokenMaskOrRevert(token);
(, lt) = _collateralTokenByMask({tokenMask: tokenMask, calcLT: true
    ↪ }); // U:[CM-42]
```

CVF-56. FIXED

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Description This assignment should be made only when “calcLT” is true.

Client Comment *A check for calcLT was added before reading ltUnderlying.*

```
1228 liquidationThreshold = ltUnderlying; // U:[CM-35]
```

CVF-57. INFO

- **Category** Flaw
- **Source** CreditManagerV3.sol

Description There are no validity checks for the “flag” argument, so it is possible to modify several flags at once, no flags at all, or non-existing flags.

Recommendation Consider adding appropriate checks.

Client Comment *This function is used internally by Gearbox contracts and is not intended for external use, so it’s unlikely that a malformed flag would be passed. It is also invoked fairly frequently, so spending extra gas on input validation is not justified.*

```
1335 function setFlagFor(address creditAccount, uint16 flag, bool value)
```

```
1349 function _enableFlag(address creditAccount, uint16 flag) internal {
```

```
1354 function _disableFlag(address creditAccount, uint16 flag) internal {
```

CVF-58. FIXED

- **Category** Unclear behavior
- **Source** BotListV3.sol

Description There are no validity checks for these arguments.

Recommendation Consider adding appropriate checks.

Client Comment *The first function parameter is now sanitized in CreditFacadeV3.setBotPermissions, while the remaining two no longer exist since payments were removed completely from BotListV3.*

```
92 uint192 permissions,
   uint72 fundingAmount,
   uint72 weeklyFundingAllowance
```

CVF-59. FIXED

- **Category** Unclear behavior
- **Source** GaugeV3.sol

Recommendation The ”SetFrozenEpoch” event should be emitted here for consistency.

Client Comment *Now event is emitted.*

```
93 +epochFrozen = true; // U:[GA-01]
```

CVF-60. FIXED

- **Category** Documentation
- **Source** CreditFacadeV3.sol

Description The semantics of the ”currentMask” argument is confusing.

Recommendation Consider explaining it better and maybe renaming the function to emphasize that it gets the inverted mask only when it is not yet known.

Client Comment *It is explained/named a bit better now.*

```
1682 +function _getInvertedQuotedTokensMask(uint256 currentMask) internal
       ↪  view returns (uint256) {
```

CVF-61. FIXED

- **Category** Bad naming
- **Source** CreditFacadeV3.sol

Description The semantics of the ”priceOracle” argument is confusing.

Recommendation Consider explaining it better and maybe renaming the function to emphasize that it gets the price oracle only when it is not yet known.

Client Comment *It is explained/named a bit better now.*

```
1717 +function _getPriceOracle(address priceOracle) internal view returns
       ↪  (address) {
```

CVF-62. FIXED

- **Category** Overflow/Underflow
- **Source** CreditManagerV3.sol

Description This leads to underflow in case offset > len.

Recommendation Consider returning an empty array in such a case.

Client Comment *Now an empty array is returned.*

```
1874 +uint256 resultLen = offset + limit > len ? len - offset : limit;
```

CVF-63. FIXED

- **Category** Suboptimal
- **Source** BotListV3.sol

Recommendation The ”approvedCreditManager” mapping is redundant, as the ”approvedCreditManagers” address set already has similar mapping inside.

Client Comment *Since bot’s forbidden status is now global, ”forbidBotEverywhere” is removed and there’s no need in a set of active credit managers, so we kept the mapping.*

```
76 +mapping(address => bool) public override approvedCreditManager;
```

```
81 +EnumerableSet.AddressSet internal approvedCreditManagers;
```

CVF-64. FIXED

- **Category** Documentation
- **Source** BotListV3.sol

Description The first key is not documented.

```
93 +/// @dev Mapping from credit account to the set of bots with non-
      ↪ zero permissions
   +mapping(address => mapping(address => EnumerableSet.AddressSet))
      ↪ internal activeBots;
```

CVF-65. INFO

- **Category** Unclear behavior
- **Source** PriceOracleV3.sol

Description This call shouldn't be performed in case the token isn't trusted and the reserve feed for the token is zero.

Client Comment *True, although we expect it not to be a big issue since most price feeds will have reserve ones.*

```
159 +(price, scale) = _getPrice(priceFeed, stalenessPeriod, skipCheck,
       ↪ decimals); // U:[PO-11]
```

9 Moderate Issues

CVF-66. FIXED

- **Category** Flaw
- **Source** QuotasLogic.sol

Recommendation This doesn't seem correct, as if "requestedChange" is negative, then it should be allowed.

Client Comment *This function is only invoked for positive 'requestedChange' values. See* Gearbox-protocol.

CVF-67. FIXED

- **Category** Flaw
- **Source** QuotasLogic.sol

Description This formula doesn't support negative "requestedChange" values.

Recommendation If negative values are not supposed to be supported, then consider changing the type of the "requestedChange" argument to "uint96".

Client Comment *Same as above*.

```
76 return uint96(requestedChange) > maxQuotaCapacity ? int96(
      ↪ maxQuotaCapacity) : requestedChange; // I:[CMQ-08,10]
```

CVF-68. FIXED

- **Category** Flaw
- **Source** ControllerTimelockV3.sol

Description The values “maxDebtCurrent” and “minDebtCurrent” were correct when transactions were enqueued and could differ from the values at the transaction execution times.

Recommendation Consider the following scenario: 1. The current debt limits are: minDebt=100, maxDebt=200. 2. Admin wants to set the limits to 50, 400 respectively. 3. Admin calls setMinDebtLimit(50) and setMaxDebtLimit(400) 4. Effectively, the scheduled transaction are: setLimits(50, 200) and setLimits(100, 400). 5. Regardless of the transaction execution order, the final outcome would not be what the admin desired. Consider obtaining the old values at the transaction execution times.

Client Comment *Same as above.*

```
144 data: abi.encode(minDebt, maxDebtCurrent)
```

```
165 data: abi.encode(minDebtCurrent, maxDebt)
```

CVF-69. INFO

- **Category** Suboptimal
- **Source** ReentrancyGuardTrait.sol

Description Using the “uint8” type for this variable make it more expensive to access, as it could share storage slot with other variables.

Recommendation Consider using the “uint256” type.

Client Comment *As far as we are aware, this is exactly opposite - a uint8 variable will be packed with other variables with sub-32 byte size into a single slot, which will make both read and write access cheaper if batched variables are accessed within the same call.*

```
12 uint8 internal _reentrancyStatus = NOT_ENTERED;
```

CVF-70. INFO

- **Category** Overflow/Underflow
- **Source** CreditLogic.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, but some intermediary calculation overflows.

Recommendation Consider using the “muldiv” function as described here: https://xn–2-mb.com/21/muldiv/ or other approaches that prevent phantom overflows.

Client Comment *Introducing muldivs into the codebase would introduce a lot of gas overhead, and also would require a lot of additional work to test and verify that arithmetic does not break as a result of the change.*
At the same time, in a large portion of highlighted cases the multiplied values have predictable sizes - they are either strictly defined, e.g., fee percentages, or very hard to make sufficiently large to overflow in practice, e.g., balances or interest indexes, as long as the protocol does not use tokens with extremely large supplies as underlyings.
At the same time, with solidity 0.8 implicit overflow checks, the phantom overflow will cause a revert, which may actually be desirable - it could prevent a potential attacker from successfully performing an operation with unreasonable manipulated values.
We have asked the auditing team for clarification on whether these overflows can transpire in realistic scenarios and whether there is actually any danger to the protocol should they occur. The auditing team has not provided a compelling example, so we believe that this change is unjustified.

```
38  return value * (block.timestamp - timestampLastUpdate) /
       ↪ SECONDS_PER_YEAR;
```

```
50  return (amount * cumulativeIndexNow) / cumulativeIndexLastUpdate -
       ↪ amount; // U:[CL-1]
```

```
101 uint256 totalFunds = totalValue * liquidationDiscount /
       ↪ PERCENTAGE_FACTOR;
```

```
103 amountToPool += totalValue * feeLiquidation / PERCENTAGE_FACTOR;
```

```
174     (ltInitial * (timestampRampEnd - block.timestamp) + ltFinal * (
           ↪ block.timestamp - timestampRampStart))
            / (timestampRampEnd - timestampRampStart)
```

```
203     (cumulativeIndexNow * newDebt * INDEX_PRECISION)
            / ((INDEX_PRECISION * cumulativeIndexNow * debt) /
               ↪ cumulativeIndexLastUpdate + INDEX_PRECISION * amount)
```

(262, 275, 295, 307, 320)

CVF-71. INFO

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Description This enforces one target contract per adapter, which seems suboptimal, as there could be several instances of the same protocol, and all such instances in theory could be handled by a single adapter instance.

Client Comment *The one contract / one adapter paradigm was deliberately chosen for compartmentalization: 1) It simplifies adapter logic (with several target contracts, an adapter would need some way to switch, which would greatly increase internal logic complexity, interface complexity and attack surface) 2) It simplifies monitoring, testing and routing (as adapters become easy to associate with their respective contracts) 3) It allows to handle any target contract idiosyncracies without affecting interactions with other target contracts*
Changing this logic would essentially require overhauling most of our codebase, and we do not see a compelling reason to do so.

```
150 mapping(address => address) public override adapterToContract;
```

CVF-72. INFO

- **Category** Readability
- **Source** CreditManagerV3.sol

Description Using assembly for manipulating with struct fields saves little gas, but makes code much harder to read and much more error prone.

Recommendation Consider using plain Solidity instead.

Client Comment *Based on our tests, using assembly to optimize storage access saves hundreds or, in some cases, thousands of gas per instance compared to plain Solidity with optimizer.*

254
```
assembly {
    let slot := add(newCreditAccountInfo.slot, 4)
    let value := or(shl(80, onBehalfOf), shl(16, number()))
    sstore(slot, value)
}
```

260
```
if (supportsQuotas) {
    //     newCreditAccountInfo.cumulativeQuotaInterest = 1;
    //     newCreditAccountInfo.quotaFees = 0;
    assembly {
        let slot := add(newCreditAccountInfo.slot, 2)
        sstore(slot, 1)
    } // U:[CM-6]
}
```

CVF-73. INFO

- **Category** Suboptimal
- **Source** PriceOracleV3.sol

Description Zero decimals is quite a normal for an ERC20 token.

Recommendation Consider supporting zero decimals.

Client Comment *Not planning to add zero-decimals tokens to the system in the foreseeable future.*

91
```
if (decimals == 0) revert PriceFeedDoesNotExistException();
```

178
```
if (decimals == 0) revert PriceFeedDoesNotExistException(); // U:[PO
    ↪ -7]
```

CVF-74. INFO

- **Category** Flaw
- **Source** PriceOracleV3.sol

Description In case a “try” call executes successfully, but the data returned from the call cannot be parsed, transaction will be reverted with no error message.

Recommendation Consider using the “call” function or even an assembly block instead of “try/catch”, to reliable handle contracts that don’t properly implemented the function being called.

Client Comment *Acknowledged, but the functionality that only allows to filter out contracts that implement the interface properly but return inappropriate values works sufficiently well for us.*

211
```
try ERC20(token).decimals() returns (uint8 _decimals) {
```

214
```
} catch {
    revert IncorrectTokenContractException(); // U:[PO-4]
}
```

CVF-75. INFO

- **Category** Overflow/Underflow
- **Source** LinearInterestRateModelV3.sol

Description Phantom overflow is possible here.

Recommendation Consider using the “muldiv” function.

Client Comment *See CVF-70*

```
118 uint256 U_WAD = (WAD * (expectedLiquidity - availableLiquidity)) /
      ↪ expectedLiquidity; // U:[LIM-3]
```

```
127     return R_base_RAY + ((R_slope1_RAY * U_WAD) / U_1_WAD); // U:[
          ↪ LIM-3]
```

```
137     return R_base_RAY + R_slope1_RAY + (R_slope2_RAY * (U_WAD -
          ↪ U_1_WAD)) / (U_2_WAD - U_1_WAD); // U:[LIM-3]
```

```
152 return R_base_RAY + R_slope1_RAY + R_slope2_RAY + (R_slope3_RAY * (
      ↪ U_WAD - U_2_WAD)) / (WAD - U_2_WAD); // U:[LIM-3]
```

```
178     uint256 U_WAD = (WAD * (expectedLiquidity - availableLiquidity))
          ↪  / expectedLiquidity; // U:[LIM-3]
```

```
180     return (U_WAD < U_2_WAD) ? ((U_2_WAD - U_WAD) *
          ↪ expectedLiquidity) / WAD : 0; // U:[LIM-3]
```

CVF-76. FIXED

- **Category** Suboptimal
- **Source** PoolQuotaKeeperV3.sol

Recommendation Should be: if (quoted > 1 && newQuoted <= 1)

Client Comment *Check for ‘quoted > 1‘ was added.*

```
184 if (newQuoted <= 1) {
```

CVF-77. INFO

- **Category** Suboptimal
- **Source** PoolQuotaKeeperV3.sol

Description This function always tries to update all the quota tokens and reverts in case update fails for any of the tokens. This is a dangerous approach. It is possible that block gas limit won't be enough to update all the tokens. Also, it is possible, that some of the tokens is broken and cannot be updated.

Recommendation Consider implementing an ability to select which tokens to update.

Client Comment *This will only happen if more than 6000 tokens are added, which is sufficiently large. There is no need to complicate this logic.*

```
417 function updateRates()
```

CVF-78. FIXED

- **Category** Suboptimal
- **Source** PoolQuotaKeeperV3.sol

Description Here, that "lastQuotaRateUpdate" value is set without actually updating the quota rate.

Recommendation Consider either updating the quota rates or clearly explaining why setting the last update timestamp without performing an update is fine.

Client Comment *The line was removed.*

```
468 lastQuotaRateUpdate = uint40(block.timestamp); // U:[PQK-8]
```

CVF-79. INFO

- **Category** Suboptimal
- **Source** PoolV3.sol

Description Calculating the borrow rate as a function of liquidity and expected liquidity doesn't make much sense from business point of view, as the optimal borrow rate is a function of market conditions, and when the borrow rate is close to optimal, liquidity will approach zero.

Recommendation Consider impoementing a statefull algorithm for discoverng the optimal borrow rate.

Client Comment *Discussed at length during calls - this out of scope of the current Gearbox V3 project.*

551
```
_baseInterestRate = IInterestRateModelV3(interestRateModel).
    ↪ calcBorrowRate({
     expectedLiquidity: expectedLiquidity_ + (supportsQuotas ?
        ↪ _calcQuotaRevenueAccrued() : 0),
     availableLiquidity: availableLiquidity_,
     checkOptimalBorrowing: checkOptimalBorrowing
}).toUint128(); // U:[P4-16]
```

CVF-80. INFO

- **Category** Suboptimal
- **Source** PoolV3.sol

Description These results are not necessarily reciprocal due to rounding errors.

Recommendation Consider calculating the amount with fee while taking rounding into account.

Client Comment *There is no need to ensure reciprocity, as these functions are never used sequentially.*

```
720 function _amountWithWithdrawalFee(uint256 amount) internal view
       ↪ returns (uint256) {
        return amount * PERCENTAGE_FACTOR / (PERCENTAGE_FACTOR -
           ↪ withdrawFee);
    }

    /// @dev Returns amount of token that would actually be sent to the
       ↪ receiver when withdrawing `amount`
    function _amountMinusWithdrawalFee(uint256 amount) internal view
       ↪ returns (uint256) {
        return amount * (PERCENTAGE_FACTOR - withdrawFee) /
           ↪ PERCENTAGE_FACTOR;
```

CVF-81. INFO

- **Category** Suboptimal
- **Source** USDTFees.sol

Recommendation This calculation rounds down, which means, that the resulting gross amount could be not enough to produce the desired net amount. Should round up.

Client Comment *According to our tests ([UTT-01,02]), this does not happen.*

```
18 uint256 amountWithBP = (amount * PERCENTAGE_FACTOR) / (
      ↪ PERCENTAGE_FACTOR - basisPointsRate); // U:[UTT_01]
```

CVF-82. INFO

- **Category** Unclear behavior
- **Source** QuotasLogic.sol

Description This function seems to calculate simple, rather than compound percentage. It this intentional?

Client Comment *Yes.*

```
20 function cumulativeIndexSince(uint192 cumulativeIndexLU, uint16 rate
      ↪ , uint256 lastQuotaRateUpdate)
```

CVF-83. INFO

- **Category** Suboptimal
- **Source** ControllerTimelockV3.sol

Description This contract allows executing queued transactions out of order or executing only some of them. This could lead to security problems. Consider the following scenario: In order to fix some problem, two transactions: A and B ought to be executed in particular order: A then B. In case the order is reversed (B then A) or the transaction B is executed, but A is skipped, the problem is not solved and even becomes worse. With the current implementation, admin may enqueue both transaction in the proper order, but then executed them in incorrect order or execute just B, but not A.

Recommendation Consider requiring enqueued transactions to be executed in the order they were enqueued and forbid skipping enqueued transactions. If a transaction is cancelled or expired, all the subsequent transaction should also be cancelled.

Client Comment *Ability to cherry pick executed transactions is actually preferable. The contract operator will not have sufficient power in practice to benefit from reordering queued transaction, in any case.*

```
27 contract ControllerTimelockV3 is PolicyManagerV3,
      ↪ IControllerTimelockV3 {
```

CVF-84. FIXED

- **Category** Suboptimal
- **Source** ControllerTimelockV3.sol

Description The “current” value passed to a “_checkPolicy” call is current at the time when a transaction is being enqueued, but it can differ from the value at the transaction execution time.

Recommendation Consider applying policies when transactions are executed, not when they are enqueued.

Client Comment *A check was implemented that the existing parameter value does not change between the transaction being enqueued and executed. This solves the primary problem of the controller being able to accidentally overwrite changes made by other parties, and also addresses this issue - since the parameter value stays the same until execution, the same policy check still holds, even though it is not performed explicitly.*

```
75     !_checkPolicy(creditManager, "EXPIRATION_DATE", uint256(
          ↪ oldExpirationDate), uint256(expirationDate))
```

```
95  if (!_checkPolicy(priceFeed, "LP_PRICE_FEED_LIMITER",
       ↪ currentLowerBound, lowerBound)) {
```

```
113    !_checkPolicy(
           creditManager, "MAX_DEBT_PER_BLOCK_MULTIPLIER", uint256(
              ↪ currentMultiplier), uint256(multiplier)
       )
```

```
137 if (!_checkPolicy(creditManager, "MIN_DEBT", uint256(minDebtCurrent)
       ↪ , uint256(minDebt))) {
```

```
158 if (!_checkPolicy(creditManager, "MAX_DEBT", uint256(maxDebtCurrent)
       ↪ , uint256(maxDebt))) {
```

```
180        !_checkPolicy(creditManager, "CREDIT_MANAGER_DEBT_LIMIT",
              ↪ uint256(debtLimitCurrent), uint256(debtLimit))
```

```
198        !_checkPolicy(creditManager, "CREDIT_MANAGER_DEBT_LIMIT",
              ↪ uint256(debtLimitCurrent), uint256(debtLimit))
```

```
230    !_checkPolicy(policyHash, uint256(ltCurrent), uint256(
          ↪ liquidationThresholdFinal)) || rampDuration < 7 days
```

```
272 if (!_checkPolicy(policyHash, uint256(oldLimit), uint256(limit))) {
```

(294, 313, 328, 348, 372)

CVF-85. FIXED

- **Category** Overflow/Underflow
- **Source** ControllerTimelockV3.sol

Description Overflow is posible here.

Recommendation Consider using safe conversion.

Client Comment *This code was removed along with all trackTotalDebt() logic*

```
190 data: abi.encode(uint128(debtLimit))
```

CVF-86. INFO

- **Category** Overflow/Underflow
- **Source** ControllerTimelockV3.sol

Description Overflow is possible here.

Recommendation Consider using safe conversion.

Client Comment *This can only overflow if the governance sets an extremely large 'delay' parameter, which is unlikely to happen.*

```
396 eta: uint40(eta),
```

```
407 eta: uint40(eta)
```

CVF-87. FIXED

- **Category** Flaw
- **Source** CreditFacadeV3.sol

Description There is no check to ensure that the active credit account isn't set already.

Recommendation Consider adding such check.

Client Comment *Added a check for creditAccount not being equal to address(1).*

```
883 ICreditManagerV3(creditManager).setActiveCreditAccount(creditAccount
    ↪ ); // F:[FA-26]
```

CVF-88. INFO

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Description This check works correctly only when “permission” has exactly one bit set.

Recommendation Consider either explicitly requiring exactly one bit to be set in “permission” or changing the check logic to: if (flags & permission == permission) ... to ensure that all the bits set in “permission” are also set in ”flags”. This would allow checking several permissions at once.

Client Comment *This is used only to check individual permissions on each ‘_multicall‘ loop that are passed as constants. The current fucntion is sufficient for its purposes.*

```
896 if (flags & permission == 0) {
```

CVF-89. FIXED

- **Category** Documentation
- **Source** CreditFacadeV3.sol

Description In some cases, this function actually may erase permissions for all bots.

Recommendation Consider either removing this logic or clearly documenting it.

Client Comment *The function no longer erases permissions automatically.*

```
1042 /// @notice Sets permissions and funding parameters for a bot
```

CVF-90. FIXED

- **Category** Flaw
- **Source** CreditFacadeV3.sol

Description The returned value is ignored.

Recommendation Consider explicitly checking that the returned value is true.

Client Comment *WETH transfer call was wrapped into safeTransfer.*

```
1214 IWETH(weth).transfer(msg.sender, msg.value); // U:[FA-7]
```

CVF-91. FIXED

- **Category** Flaw
- **Source** CreditConfiguratorV3.sol

Description The forbidden tokens are actually not copied from the existing credit configurator.

Client Comment *Comment removed.*

```
115 // 3. Forbidden token set stores all forbidden tokens - used for
       ↪ parameter migration when changing the Credit Facade
```

CVF-92. INFO

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Description Starting ramp later than desired, but preserving the duration could make the ramp to end later than needed.

Recommendation Consider just reverting in case “rampStart” is in the past.

Client Comment *This is acceptable - timestamp ramp periods need not be exact (as long as duration is preserved), and reverting on ramp start being in the past would greatly complicate execution from a timelock.*

```
286 rampStart = block.timestamp > rampStart ? uint40(block.timestamp) :
       ↪ rampStart; // I:[CC-30]
```

CVF-93. FIXED

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Recommendation Should be “rampStart + rampDuration”.

Client Comment *Changed to rampStart + rampDuration.*

```
306 timestampRampEnd: uint40(block.timestamp) + rampDuration
```

CVF-94. INFO

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Description This is an unexpected side effect of the function.

Recommendation Consider just checking that liquidation thresholds for all token are below the new underlying LT, and reverting in case this rule is violated. Another way to avoid this side efeict is to allow a stored token LT to be higher than underlying LT, but use min(tokenLT, underlyingLT) as the effective token LT.

Client Comment *The side effect is anticipated and accounted for, as changing the underlying LT is a rare and major change.*

```
618 // An LT of an ordinary collateral token cannot be larger than the
       ↪ LT of underlying
    // As such, all LTs need to be checked and reduced if needed
620 // NB: This action will interrupt all ongoing LT ramps
```

CVF-95. INFO

- **Category** Flaw
- **Source** CreditConfiguratorV3.sol

Description This check doesn't seem reliable.

Recommendation In order to make credit configuration change procedure safer, consider implementing a two step schema: the old configurator suggests the new configurator and then the new configurator accepts the transition. This would guarantee that the new configurator is alive.

Client Comment *This would be incredibly cumbersome, as all configuration is subject to a timelock. The current system is sufficient for us.*

```
759 _revertIfContractIncompatible(_creditConfigurator); // I:[CC-20]
```

CVF-96. FIXED

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Description In case the liquidator is already added into the credit facade, it will not be added into the “emergencyLiquidatorSet”.

Recommendation Consider always adding to the “emergencyLiquidatorSet”.

Client Comment *The set update was moved to before early return.*

```
932 if (cf.canLiquidateWhilePaused(liquidator)) return;
```

```
935 emergencyLiquidatorsSet.add(liquidator); // I:[CC-27]
```

CVF-97. FIXED

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Description In case the liquidator not added into the credit facade, it will not be removed from the “emergencyLiquidatorSet”.

Recommendation Consider always removing from the “emergencyLiquidatorSet”.

Client Comment *The set update was moved to before early return.*

```
949 if (!cf.canLiquidateWhilePaused(liquidator)) return;
```

```
952 emergencyLiquidatorsSet.remove(liquidator); // I:[CC-28]
```

CVF-98. FIXED

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Description Despite the comment, the block number is not truncated to 64 bits.

Recommendation Consider truncating.

Client Comment *The block number is now truncated on writing into the slot.*

```
251 // newCreditAccountInfo.since = uint64(block.number); // U:[CM-6]
```

```
256     let value := or(shl(80, onBehalfOf), shl(16, number()))
```

CVF-99. INFO

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Description Handling failed transfers in such way is a bad idea, as a failed transfer could consume too much gas, so the remaining fas wouldn't be enough to add withdrawal.

Recommendation Consider implementing an ability to forcefully add withdrawal without trying direct transfer first.

Client Comment *We have not seen any production examples of tokens that exhibit such behavior, aside from this being a distinct possibility for tokens that implement post-transfer hooks. However, such tokens are explicitly prohibited in Gearbox for various security considerations, so we do not believe that any changes are warranted.*

1155
```
///         If transfer fails (e.g., `to` gets blacklisted in the
   ↪ token), the token will be transferred
///         to withdrawal manager from which `to` can later claim it
   ↪  to an arbitrary address.
```

CVF-100. FIXED

- **Category** Flaw
- **Source** BotListV3.sol

Description Here implicit underflow check performed by compiler is used to enforce and important business-level constraint. This is a bad practice, as it make code harder to read and more error prone. Also, it uses assert rather than revert which is less efficient.

Recommendation Consider explicitly checking business-level constraints and using implicit check as a second line of defence.

Client Comment *Explicit checks were added.*

183
```
bf.remainingWeeklyAllowance -= totalAmount; // F: [BL-05]
bf.remainingFunds -= totalAmount; // F: [BL-05]
```

186
```
balanceOf[payer] -= totalAmount; // F: [BL-05]
```

211
```
balanceOf[msg.sender] -= amount; // F: [BL-04]
```

CVF-101. INFO

- **Category** Overflow/Underflow
- **Source** CreditManagerV3.sol

Recommendation Overflow is possible, consider using safe conversion.

Client Comment *That would take quite a bit even on the chains with block times smaller than on mainnet.*

```
709 +currentCreditAccountInfo.lastDebtUpdate = uint64(block.number); //
       ↪ U:[CM-10, 11]
```

CVF-102. FIXED

- **Category** Overflow/Underflow
- **Source** BalancesLogic.sol

Description Overflow is possible when converting types.

Recommendation Consider using safe conversion.

Client Comment *Now uses safe conversion.*

```
55 +expected[i] = Balance({token: deltas[i].token, balance: (int256(
       ↪ balance) + deltas[i].amount).toUint256()}); // U:[BLL-1]
```

CVF-103. FIXED

- **Category** Unclear behavior
- **Source** GearStakingV3.sol

Description Thee functions don't ensure bidirectional one-to-one relationship between a successor and migrator.

Recommendation Consider implementing some logic to ensure this.

Client Comment *Now "setSuccessor" checks that "newSuccessor" has current contract set as migrator (checking for one-to-one relationship seems redundant).*

```
385 +function setSuccessor(address newSuccessor) external override
       ↪ configuratorOnly {
```

```
397 +function setMigrator(address newMigrator) external override
       ↪ configuratorOnly {
```

CVF-104. FIXED

- **Category** Flaw
- **Source** CreditManagerV3.sol

Description There is no check to ensure that ”tokenMask” has exactly one bit set.

Recommendation Consider checking like this: require (tokenMask & tokenMask - 1 == 0);

Client Comment *It can’t be used to harm the system (”getTokenByMask” would revert), but we’ve added the suggested explicit check in ”fullCollateralCheck” anyways.*

```
1291 +tokenMask = collateralHints[hintsIdx++];
```

CVF-105. FIXED

- **Category** Suboptimal
- **Source** AccountFactoryV3.sol

Description The ”FactoryParams.tail” field is ”uint40” while the indexed array has only 2^32 elements.

Recommendation Consider making these values consistent. Also, an index range check is performed here by compiler. Consider avoiding it by using assembly.

Client Comment *We decided to use a mapping here after all.*

```
105 +_queuedAccounts[msg.sender][fp.tail] =
```

CVF-106. FIXED

- **Category** Suboptimal
- **Source** BotListV3.sol

Description This function sets forbidden flag for each approved credit manager, which has two drawbacks: i) it doesn't affect credit managers approved after the call, and ii) it overwrites individual flags, making the operation irreversible. Instead of setting individual flag, consider setting a global per-bot "forbidden" flag.

Recommendation Consider a bot to be forbidden for a credit manager when either the global flag for the bot is set, or the individual flag for this bot and credit manager is set. This would also make this function to scale better.

Client Comment *Bot's forbidden status is now global.*

```
398 +/// @notice Sets the bot's forbidden status in all credit managers
    +function setBotForbiddenStatusEverywhere(address bot, bool status)
       ↪ external override configuratorOnly {
```

CVF-107. INFO

- **Category** Unclear behavior
- **Source** BalancesLogic.sol

Description In the case "comparison" is neither "GREATER" nor "LESS", this code silently returns false.

Recommendation Consider reverting in such a case.

Client Comment *Comparison currently can't be anything but "GREATER" or "LESS".*

```
48 +return (comparison == Comparison.GREATER && current >= value)
   +    || (comparison == Comparison.LESS && current <= value); // U:[
      ↪ BLL-1]
```

CVF-108. INFO

- **Category** Suboptimal
- **Source** BalancesLogic.sol

Description The ”comparison” value is passed into every call and is checked there, which is suboptimal.

Recommendation Consider implementing two versions of the loop and using the ”comparison” value only once to chooses between these two versions.

Client Comment *That might harm readability and doesn't save that much gas to implement it at this stage.*

97
```
+if (!BalancesLogic.checkBalance(creditAccount, balances[i].token,
   ↪ balances[i].balance, comparison)) {
```

CVF-109. INFO

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Recommendation It would be more reasonable to use ”amount == 0” as a special value meaning ”whole balance” and withdrawing zero amount doesn't make sense.

Client Comment *Arguable, many projects use type's maximum to indicate full amount, and this seems more reasonable to us.*

1052
```
+if (amount == type(uint256).max) {
```

CVF-110. INFO

- **Category** Unclear behavior
- **Source** PriceOracleV3.sol

Description This call should be performed only when ”useReserve” is false.

Client Comment *All this data is stored in the same slot, so it won't change much.*

130
```
+(priceFeed, stalenessPeriod, skipCheck, decimals, useReserve,
   ↪ trusted) = _getPriceFeedParams(token);
```

CVF-111. INFO

- **Category** Procedural
- **Source** GearStakingV3.sol

Description Ignoring errors is a bad practice that could hide problems.

Recommendation Consider reverting in case the ”permit” call failed.

Client Comment Openzeppelin contracts

```
90 +try IERC20Permit(gear).permit(msg.sender, address(this), amount,
      ↪ deadline, v, r, s) {} catch {} // U:[GS-02]
```

CVF-112. INFO

- **Category** Suboptimal
- **Source** BotListV3.sol

Description This loop doesn’t scale and could exceed the block gas limit.

Recommendation Consider implementing an ability to split it into several transactions.

Client Comment *The restriction on active bots count is implemented in the credit facade. Also, the function is only called when closing an account which is no longer required.*

```
366 +for (uint256 i = accountBots.length(); i != 0; --i) {
```

10 Minor Issues

CVF-113. INFO

- **Category** Procedural
- **Source** PriceOracleV3.sol

Recommendation Consider specifying as “^0.8.0” unless there is something special about this particular version.

```
4 pragma solidity ^0.8.17;
```

CVF-114. INFO

- **Category** Bad datatype
- **Source** PriceOracleV3.sol

Recommendation The key type for this mapping should be “IERC20”.

Client Comment *We decided that it’s more convenient for us not to use interfaces in state variables and function arguments.*

```
33 mapping(address => PriceFeedParams) internal _priceFeedsParams;
```

CVF-115. INFO

- **Category** Procedural
- **Source** PriceOracleV3.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
37 constructor(address addressProvider) ACLNonReentrantTrait(
   ↪ addressProvider) {}
```

CVF-116. INFO

- **Category** Overflow/Underflow
- **Source** PriceOracleV3.sol

Description Phantom overflow is possible here.

Recommendation Consider using the “muldiv” function.

Client Comment *Expected behavior; under normal function the amount in numerator is dollar value with <= 26 decimals, so we’d prefer it to overflow and revert when it gets huge for some reason.*

```
57 return amount * price / scale; // U:[PO-9]
```

```
64 return amount * scale / price; // U:[PO-9]
```

CVF-117. INFO

- **Category** Bad datatype
- **Source** PriceOracleV3.sol

Recommendation The type of the “priceFeed” argument should be “AggregatorV3Interface”.

Client Comment *We decided that it’s more convenient for us not to use interfaces in state variables and function arguments.*

```
98 function _getPrice(address priceFeed, uint32 stalenessPeriod, bool
   ↪ skipCheck, uint8 decimals)
```

```
147 function setPriceFeed(address token, address priceFeed, uint32
    ↪ stalenessPeriod)
```

```
170 function setReservePriceFeed(address token, address priceFeed,
    ↪ uint32 stalenessPeriod)
```

CVF-118. FIXED

- **Category** Suboptimal
- **Source** PriceOracleV3.sol

Recommendation Using a free memory pointer is redundant here. Just use scratch space instead: https://docs.soliditylang.org/en/v0.8.17/assembly.html?highlight=scratch#memory-management

Client Comment *Fixed as recommended.*

```
136 let ptr := mload(0x40)
```

CVF-119. INFO

- **Category** Procedural
- **Source** SanityCheckTrait.sol

Recommendation Consider specifying as "^0.8.0" unless there is something special about this particular version. Also relevant for: PriceFeedValidationTrait.sol, ACLTrait.sol, ReentrancyGuardTrait.sol, ACLNonReentrantTrait.sol, ContractsRegisterTrait.sol, USDT_Transfer.sol, LinearInterestRateModelV3.sol, PoolQuotaKeeperV3.sol, PoolV3_USDT.sol, PoolV3.sol, CreditLogic.sol, CollateralLogic.sol, CreditAccountHelper.sol, UnsafeERC20.sol, BitMask.sol, BalancesLogic.sol, QuotasLogic.sol, PolicyManagerV3.sol, ControllerTimelockV3.sol, GearStakingV3.sol, GaugeV3.sol, CreditFacadeV3.sol, CreditManagerV3_USDT.sol, CreditAccountV3.sol, CreditConfiguratorV3.sol, CreditManagerV3.sol, AddressProviderV3.sol, BotListV3.sol, AccountFactoryV3.sol.

Client Comment *We do not see any sufficient reason to downgrade the pragma.*

```
4 pragma solidity ^0.8.17;
```

CVF-120. INFO

- **Category** Procedural
- **Source** ACLTrait.sol

Description We didn't review this file.

```
6 import {IACL} from "@gearbox-protocol/core-v2/contracts/interfaces/
    ↪ IACL.sol";
```

CVF-121. INFO

- **Category** Procedural
- **Source** ReentrancyGuardTrait.sol

Description Defining top-level constants in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the constant definitions into the contract or moving them into a separate file.

Client Comment *Defining them like this simplifies their usage in tests.*

```
6 uint8 constant NOT_ENTERED = 1;
  uint8 constant ENTERED = 2;
```

CVF-122. INFO

- **Category** Procedural
- **Source** ACLNonReentrantTrait.sol

Description We didn't review this file.

```
8 import {IACL} from "@gearbox-protocol/core-v2/contracts/interfaces/
    ↪ IACL.sol";
```

CVF-123. INFO

- **Category** Bad naming
- **Source** ACLNonReentrantTrait.sol

Recommendation The error name is inaccurate, as configurator is also permitted as a caller.

Client Comment *Configurator is a root admin in the system, so it is logical that they have access to all admin functions.*

```
38 revert CallerNotControllerException();
```

CVF-124. INFO

- **Category** Procedural
- **Source** ContractsRegisterTrait.sol

Description We didn't review this file.

```
6 import {IContractsRegister} from "@gearbox-protocol/core-v2/
    ↪ contracts/interfaces/IContractsRegister.sol";
```

CVF-125. FIXED

- **Category** Documentation
- **Source** ContractsRegisterTrait.sol

Description It is unclear why version is hardcoded to one.

Recommendation Consider explaining. Also, the version number should be a named constant.

Client Comment *Fixed by changing to NO_VERSION_CONTROL6*

```
34 contractsRegister = IAddressProviderV3(addressProvider).
    ↪ getAddressOrRevert(AP_CONTRACTS_REGISTER, 1);
```

CVF-126. FIXED

- **Category** Procedural
- **Source** USDT_Transfer.sol

Recommendation This interface should be defined in a separate file named "IUSDT.sol".

Client Comment *Moved to a separate file.*

```
8 interface IUSDT {
```

CVF-127. FIXED

- **Category** Documentation
- **Source** USDT_Transfer.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

Client Comment *Format added to docstring.*

```
9 function basisPointsRate() external view returns (uint256);
```

CVF-128. FIXED

- **Category** Suboptimal
- **Source** LinearInterestRateModelV3.sol

Recommendation It would be more efficient to calculate as: x * (WAD / PERCENTAGE_FACTOR) and compiler would be able to precompute the fraction.

Client Comment *Changed per recommendation.*

```
75 U_1_WAD = (WAD * U_1) / PERCENTAGE_FACTOR; // U:[LIM-1]
   U_2_WAD = (WAD * U_2) / PERCENTAGE_FACTOR; // U:[LIM-1]
```

CVF-129. FIXED

- **Category** Suboptimal
- **Source** LinearInterestRateModelV3.sol

Recommendation It would be more efficient to calculate as: x * (RAY / PERCENTAGE_FACTOR) and compiler would be able to precompute the fraction.

Client Comment *Changed per recommendation.*

```
79 R_base_RAY = (RAY * R_base) / PERCENTAGE_FACTOR; // U:[LIM-1]
80 R_slope1_RAY = (RAY * R_slope1) / PERCENTAGE_FACTOR; // U:[LIM-1]
   R_slope2_RAY = (RAY * R_slope2) / PERCENTAGE_FACTOR; // U:[LIM-1]
   R_slope3_RAY = (RAY * R_slope3) / PERCENTAGE_FACTOR; // U:[LIM-1]
```

CVF-130. FIXED

- **Category** Suboptimal
- **Source** LinearInterestRateModelV3.sol

Recommendation This could be simplified as: if (expectedLiquidity <= availableLiquidity)

Client Comment *Changed per recommendation.*

```
110 if (expectedLiquidity == 0 || expectedLiquidity < availableLiquidity
    ↪ ) {
```

CVF-131. FIXED

- **Category** Suboptimal
- **Source** LinearInterestRateModelV3.sol

Recommendation The first part of the condition is redundant, as it is guaranteed to be true here.

Client Comment *Changed per recommendation.*

```
136 if (U_WAD >= U_1_WAD && U_WAD < U_2_WAD) {
```

CVF-132. FIXED

- **Category** Suboptimal
- **Source** LinearInterestRateModelV3.sol

Recommendation It would be more efficient to calculate as: x / (WAD / PERCENTAGE_FACTOR) as compiler will be able to precompute the denominator.

Client Comment *Changed per recommendation.*

```
161 U_1 = uint16((U_1_WAD * PERCENTAGE_FACTOR) / WAD); // U:[LIM-1]
    U_2 = uint16((U_2_WAD * PERCENTAGE_FACTOR) / WAD); // U:[LIM-1]
```

CVF-133. FIXED

- **Category** Suboptimal
- **Source** LinearInterestRateModelV3.sol

Recommendation It would be more efficient to calculate as: x / (RAY / PERCENTAGE_FACTOR) as compiler will be able to precompute the denominator.

Client Comment *Changed per recommendation.*

163
```
R_base = uint16(R_base_RAY * PERCENTAGE_FACTOR / RAY); // U:[LIM-1]
R_slope1 = uint16(R_slope1_RAY * PERCENTAGE_FACTOR / RAY); // U:[LIM
    ↪ -1]
R_slope2 = uint16(R_slope2_RAY * PERCENTAGE_FACTOR / RAY); // U:[LIM
    ↪ -1]
R_slope3 = uint16(R_slope3_RAY * PERCENTAGE_FACTOR / RAY); // U:[LIM
    ↪ -1]
```

CVF-134. FIXED

- **Category** Overflow/Underflow
- **Source** PoolQuotaKeeperV3.sol

Description Phantom overflow is possible here.

Recommendation Consider calculating in 256 bits and then performing safe conversion to 128 bits.

Client Comment *Prevented by casting quotaChange to uint256 before multiplication.*

161
```
fees = uint128(uint96(quotaChange)) * quotaIncreaseFee /
    ↪ PERCENTAGE_FACTOR; // U: [PQK-15]
```

CVF-135. FIXED

- **Category** Procedural
- **Source** PoolQuotaKeeperV3.sol

Description The expression "-quotaChange" is calculated twice.

Recommendation Consider calculating once and reusing.

Client Comment *Changed per recommendation.*

180
```
newQuoted = quoted - uint96(-quotaChange);
tokenQuotaParams.totalQuoted -= uint96(-quotaChange); // U: [PQK-15]
```

CVF-136. FIXED

- **Category** Suboptimal
- **Source** PoolQuotaKeeperV3.sol

Description The only case how the first condition could be false but the second condition be true is quoted == 1 which means that the assignment would do nothing.

Recommendation Consider refactoring as: if (quoted > 1) { ... accountQuota.quota = 1; }

Client Comment *Changed per recommendation.*

227
```
if (quoted > 1) {
```

242
```
if (quoted != 0) {
    accountQuota.quota = 1; // U: [PQK-16]
}
```

CVF-137. FIXED

- **Category** Flaw
- **Source** PoolQuotaKeeperV3.sol

Description There is no range check for the "fee" argument.

Recommendation Consider adding an appropriate check.

Client Comment *Added a check for [0, PERCENTAGE_FACTOR] range.*

521
```
function setTokenQuotaIncreaseFee(address token, uint16 fee)
```

CVF-138. INFO

- **Category** Procedural
- **Source** PoolV3.sol

Description We didn't review this file.

```
7 import {SafeERC20} from "@1inch/solidity-utils/contracts/libraries/
    ↪ SafeERC20.sol";
```

CVF-139. INFO

- **Category** Procedural
- **Source** PoolV3.sol

Description Defining a top-level structure in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the structure definition into the contract or moving it into a separate file.

Client Comment *Defining them like this simplifies their usage in tests.*

```
42 struct DebtParams {
```

CVF-140. FIXED

- **Category** Suboptimal
- **Source** PoolV3.sol

Recommendation As this variable is immutable, it would be more efficient to implement two versions of this smart contract: one that supports quotas and another that doesn't support.

Client Comment *All functionality for supportsQuotas == false case was removed. It is now assumed that contracts always support quotas.*

```
70 bool public immutable override supportsQuotas;
```

CVF-141. FIXED

- **Category** Unclear behavior
- **Source** PoolV3.sol

Description These events are emitted even if nothing actually changed.

Client Comment *The functions now return early if the new value is the same, preventing assignment and event emission.*

```
694 emit SetWithdrawFee(newWithdrawFee); // U:[P4-26]
```

```
700 emit SetTotalDebtLimit(limit); // U:[P4-3,25]
```

CVF-142. FIXED

- **Category** Suboptimal
- **Source** CreditLogic.sol

Recommendation Should be "<=".

Client Comment *Changed per recommendation.*

```
151 if (block.timestamp < timestampRampStart) {
```

CVF-143. FIXED

- **Category** Readability
- **Source** CreditLogic.sol

Recommendation Should be "else if".

Client Comment *Changed per recommendation.*

```
154 if (block.timestamp < timestampRampStart + rampDuration) {
```

CVF-144. FIXED

- **Category** Readability
- **Source** CreditLogic.sol

Recommendation Should be “else return”.

Client Comment *Changed per recommendation.*

```
159 return ltFinal; // U:[CL-05]
```

CVF-145. FIXED

- **Category** Suboptimal
- **Source** CreditLogic.sol

Description These assignments are redundant, as they will be anyway overwritten later.

Recommendation Consider removing ithem.

Client Comment *newDebt is now assigned only once at the end of the function.*

```
282 newDebt = debt; // U:[CL-03A]
```

CVF-146. INFO

- **Category** Procedural
- **Source** CollateralLogic.sol

Description We didn’t review this file.

```
7 import {SafeERC20} from "@1inch/solidity-utils/contracts/libraries/
    ↪ SafeERC20.sol";
```

CVF-147. INFO

- **Category** Readability
- **Source** CollateralLogic.sol

Recommendation Consider explicitly initializing “i” with zero value.

Client Comment *We believe that this change would make no difference for readability.*

```
128 for (uint256 i; i < len;) {
```

```
191 for (uint256 i; tokensToCheckMask != 0;) {
```

CVF-148. FIXED

- **Category** Procedural
- **Source** CollateralLogic.sol

Recommendation This should be done inside the conditional statement above.

Client Comment *Iteration over mask itself has changed.*

```
226 tokensToCheckMask = tokensToCheckMask.disable(tokenMask); // U:[CLL
    ↪ -3]
```

CVF-149. INFO

- **Category** Procedural
- **Source** CreditAccountHelper.sol

Description We didn't review this file.

```
7 import {SafeERC20} from "@1inch/solidity-utils/contracts/libraries/
    ↪ SafeERC20.sol";
```

CVF-150. FIXED

- **Category** Suboptimal
- **Source** CreditAccountHelper.sol

Recommendation Here "abi.decode(result, (bool)) == true" could be simplified to just "abi.decode(result, (bool))".

Client Comment *Changed per recommendation.*

```
55 if (result.length == 0 || abi.decode(result, (bool)) == true) {
```

CVF-151. INFO

- **Category** Readability
- **Source** CreditAccountHelper.sol

Recommendation Should be "else return".

Client Comment *We believe that keeping the revert condition separate is better for readability.*

```
64 return false;
```

CVF-152. FIXED

- **Category** Suboptimal
- **Source** CreditAccountHelper.sol

Recommendation Conversion to “ICreditAccountBase” is redundant, as “creditAccount” is already “ICreditAccountBase”.

Client Comment *Changed per recommendation.*

```
74 ICreditAccountBase(creditAccount).safeTransfer(token, to, amount);
```

CVF-153. INFO

- **Category** Suboptimal
- **Source** BitMask.sol

Description This function is very inefficient.

Recommendation Consider optimizing like this:

```
unchecked \{
  enabledTokensMask -= (enabledTokensMask >> 1) \&0
      ↪ x5555555555555555555555555555555555555555555555555555555555555555;
  enabledTokensMask = (enabledTokensMask \& 0
      ↪ x3333333333333333333333333333333333333333333333333333333333333333) + ((
      ↪ enabledTokensMask >> 2) \&0
      ↪ x3333333333333333333333333333333333333333333333333333333333333333);
  enabledTokensMask = (enabledTokensMask + (enabledTokensMask >> 4)) \&0
      ↪ x0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F;
  totalTokensEnabled = (enabledTokensMask *0
      ↪ x0001010101010101010101010101010101010101010101010101010101010101 >> 248)
      ↪ + (enabledTokensMask \& 0xFF);
\}
```

Client Comment *While computationally elegant, this is unreadable and can possibly be less efficient for a small number of tokens. We implemented an alternative approach that also improves over the initial one.*

```
41 function calcEnabledTokens(uint256 enabledTokensMask) internal pure
     ↪ returns (uint256 totalTokensEnabled) {
```

CVF-154. FIXED

- **Category** Procedural
- **Source** USDTFees.sol

Description This comment belongs to the “amountUSDTFee” function, rather than to the whole library.

Recommendation Consider placing it accordingly.

Client Comment *Comments in the file were improved in general.*

```
10 /// @dev Computes amounts to send / receive with USDT fees accounted
      ↪  for
```

CVF-155. FIXED

- **Category** Procedural
- **Source** USDTFees.sol

Recommendation Brackets around multiplication are redundant.

Client Comment *Brackets removed.*

```
18 uint256 amountWithBP = (amount * PERCENTAGE_FACTOR) / (
      ↪ PERCENTAGE_FACTOR - basisPointsRate); // U:[UTT_01]
```

CVF-156. INFO

- **Category** Procedural
- **Source** BalancesLogic.sol

Description We didn’t review these files.

```
7 import {SafeERC20} from "@1inch/solidity-utils/contracts/libraries/
      ↪ SafeERC20.sol";
```

CVF-157. INFO

- **Category** Procedural
- **Source** BalancesLogic.sol

Description Defining a top-level struct in a file named after a library makes it harder to navigate through code.

Recommendation Consider either moving the struct definition into the library or moving it into a separate file.

Client Comment *Defining them like this simplifies their usage in tests.*

```
14 struct BalanceWithMask {
```

CVF-158. FIXED

- **Category** Procedural
- **Source** QuotasLogic.sol

Recommendation Brackets around multiplication are redundant.

Client Comment *Brackets removed.*

```
27 + (RAY_DIVIDED_BY_PERCENTAGE * (block.timestamp -
      ↪ lastQuotaRateUpdate) * rate) / SECONDS_PER_YEAR
```

CVF-159. INFO

- **Category** Procedural
- **Source** PolicyManagerV3.sol

Description Defining a top-level struct in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the struct definition into the contract or moving it into a separate file.

Client Comment *Defining them like this simplifies their usage in tests.*

```
12 struct Policy {
```

CVF-160. FIXED

- **Category** Suboptimal
- **Source** PolicyManagerV3.sol

Description These constants don't need to be public. Note, that public constants increase contact size and transaction gas cost.

Recommendation Consider declaring these constants as internal.

Client Comment *Visibility changed ot internal.*

```
54 uint256 public constant CHECK_EXACT_VALUE_FLAG = 1;
   uint256 public constant CHECK_MIN_VALUE_FLAG = 1 << 1;
   uint256 public constant CHECK_MAX_VALUE_FLAG = 1 << 2;
   uint256 public constant CHECK_MIN_CHANGE_FLAG = 1 << 3;
   uint256 public constant CHECK_MAX_CHANGE_FLAG = 1 << 4;
   uint256 public constant CHECK_MIN_PCT_CHANGE_FLAG = 1 << 5;
60 uint256 public constant CHECK_MAX_PCT_CHANGE_FLAG = 1 << 6;
```

CVF-161. INFO

- **Category** Procedural
- **Source** PolicyManagerV3.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Client Comment *Here it should be obvious that the constructor initializes the parent class, while there is no child specific code. We have not encountered any other codebases that clarify this and it seems redundant.*

```
68 constructor(address _addressProvider) ACLNonReentrantTrait(
      ↪ _addressProvider) {}
```

CVF-162. FIXED

- **Category** Unclear behavior
- **Source** PolicyManagerV3.sol

Description These functions should emit some events.

Client Comment *Functions now properly emit events.*

```
74 function setPolicy(bytes32 policyHash, Policy memory initialPolicy)
```

```
84 function disablePolicy(bytes32 policyHash)
```

```
97 function setGroup(address contractAddress, string calldata group)
     ↪ external configuratorOnly {
```

CVF-163. FIXED

- **Category** Procedural
- **Source** PolicyManagerV3.sol

Description This function allows overwriting the policy for a given policy hash.

Recommendation Consider either forbidding this behavior or clearly documenting it.

Client Comment *The function is intended to allow to overwrite policy values, so the parameter is renamed to policyParams*

```
74 function setPolicy(bytes32 policyHash, Policy memory initialPolicy)
```

CVF-164. FIXED

- **Category** Suboptimal
- **Source** PolicyManagerV3.sol

Description Here a value just written into the storage is read from the storage back.

Recommendation Consider reusing the written value.

Client Comment *Adjusted code to avoid this.*

```
149     policy.referencePoint = oldValue; // F: [PM-06]
```

```
153 referencePoint = policy.referencePoint;
```

CVF-165. FIXED

- **Category** Suboptimal
- **Source** PolicyManagerV3.sol

Description This expression is calculated twice.

Recommendation Consider calculating once and reusing.

Client Comment *The expression is now calculated once if either of PCT_DIFF flags is set.*

```
165 uint256 pctDiff = diff * PERCENTAGE_FACTOR / referencePoint;
```

```
170 uint256 pctDiff = diff * PERCENTAGE_FACTOR / referencePoint;
```

CVF-166. INFO

- **Category** Procedural
- **Source** ControllerTimelockV3.sol

Description We didn't review this file.

```
13 import {ILPPriceFeedV2} from "@gearbox-protocol/core-v2/contracts/
   ↪ interfaces/ILPPriceFeedV2.sol";
```

CVF-167. FIXED

- **Category** Bad naming
- **Source** ControllerTimelockV3.sol

Description The variable name is too generic.

Recommendation Consider making it more specific.

Client Comment *Global delay variable is removed since delays were changed to be per-olicy.*

```
38 uint256 public delay = 1 days;
```

CVF-168. INFO

- **Category** Bad datatype
- **Source** ControllerTimelockV3.sol

Recommendation The default value should be a named constant.

Client Comment *No longer relevant.*

```
38 uint256 public delay = 1 days;
```

CVF-169. INFO

- **Category** Procedural
- **Source** ControllerTimelockV3.sol

Recommendation This should be executed after range checking the “rampDuration” and “rampStart” arguments.

Client Comment *This makes the code less readable and there are no significant benefits.*

```
224 bytes32 policyHash = keccak256(abi.encode(_group[creditManager],
    ↪ _group[token], "TOKEN_LT"));
```

```
226 address creditConfigurator = ICreditManagerV3(creditManager).
    ↪ creditConfigurator();
    uint256 ltCurrent = ICreditManagerV3(creditManager).
    ↪ liquidationThresholds(token);
```

CVF-170. FIXED

- **Category** Bad datatype
- **Source** ControllerTimelockV3.sol

Recommendation The minimum ramp duration should be a named constant.

Client Comment *Introduced a constant for this.*

```
230 !_checkPolicy(policyHash, uint256(ltCurrent), uint256(
    ↪ liquidationThresholdFinal)) || rampDuration < 7 days
```

CVF-171. INFO

- **Category** Procedural
- **Source** GearStakingV3.sol

Description We didn't review this file.

```
6 import {SafeERC20} from "@1inch/solidity-utils/contracts/libraries/
    ↪ SafeERC20.sol";
```

CVF-172. INFO

- **Category** Suboptimal
- **Source** GearStakingV3.sol

Description Performing these checks on every loop iteration is suboptimal.

Recommendation Consider splitting this loop into two loops.

Client Comment *We do not believe that this will significantly increase efficiency or readability.*

```
220 if (i < epochDiff) {
```

```
225     (i + epochDiff < EPOCHS_TO_WITHDRAW) ? wd.withdrawalsPerEpoch[i
        ↪ + epochDiff] : 0;
```

CVF-173. FIXED

- **Category** Readability
- **Source** GaugeV3.sol

Recommendation Should be "to remove votes from".

Client Comment *Comments across the file were fixed.*

```
150 /// @param user User to assign votes to
```

CVF-174. FIXED

- **Category** Readability
- **Source** GaugeV3.sol

Recommendation Should be ”to subtract”.

Client Comment *Comments across the file were fixed.*

```
151 /// @param votes Amount of votes to add
    /// @param token Token to add votes to
    /// @param lpSide Side to add votes to: `true` for LP side, `false`
       ↪ for CA side
```

CVF-175. FIXED

- **Category** Readability
- **Source** GaugeV3.sol

Recommendation Should be ”to remove”.

Client Comment *Comments across the file were fixed.*

```
189 /// @param user User to assign votes to
```

CVF-176. FIXED

- **Category** Readability
- **Source** GaugeV3.sol

Recommendation Should be ”to subtract”.

Client Comment *Comments across the file were fixed.*

```
190 /// @param votes Amount of votes to add
```

CVF-177. FIXED

- **Category** Unclear behavior
- **Source** GaugeV3.sol

Description This event is emitted even if nothing actually changed.

Client Comment *Voter is no longer mutable, so the associated function does not exist.*

```
225 emit SetVoter({newVoter: newVoter}); // U:[GA-09]
```

CVF-178. INFO

- **Category** Procedural
- **Source** CreditFacadeV3.sol

Description We didn't review these files.

```
32 import {IPriceOracleV2} from "@gearbox-protocol/core-v2/contracts/
   ↪ interfaces/IPriceOracleV2.sol";
```

```
35 import {IDegenNFTV2} from "@gearbox-protocol/core-v2/contracts/
   ↪ interfaces/IDegenNFTV2.sol";
   import {IWETH} from "@gearbox-protocol/core-v2/contracts/interfaces/
   ↪ external/IWETH.sol";
```

CVF-179. INFO

- **Category** Procedural
- **Source** CreditFacadeV3.sol

Description Defining top-level constants in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the constant definitions into the contract or moving them into a separate file.

Client Comment *Defining them like this simplifies their usage in tests.*

```
45 uint256 constant OPEN_CREDIT_ACCOUNT_FLAGS = ALL_PERMISSIONS
```

```
48 uint256 constant CLOSE_CREDIT_ACCOUNT_FLAGS =
   ↪ EXTERNAL_CALLS_PERMISSION;
```

CVF-180. FIXED

- **Category** Documentation
- **Source** CreditFacadeV3.sol

Description The comment is confusing.

Recommendation Consider elaborating more.

Client Comment *Clarified that this is the maximal multiple of quota to account's maxDebt.*

```
64 /// @notice maxDebt to maxQuota multiplier
   uint256 public constant maxQuotaMultiplier = 8;
```

CVF-181. FIXED

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Description There is no access level specified for this variable, so internal access will be used by default.

Recommendation Consider explicitly specifying an access level.

Client Comment *supportsQuotas-related logic was removed, so this is no longer relevant.*

```
86 bool immutable supportsQuotas;
```

CVF-182. INFO

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Recommendation It would be more flexible to use a generic role-based authorization model, rather than separate mapping for different permissions.

Client Comment *This is the only special role tracked in CreditFacade, and does not justify inheriting a specialized access library, which will also increase contract size.*

```
122 mapping(address => bool) public override canLiquidateWhilePaused;
```

CVF-183. FIXED

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Description As the “trackTotalDebt” variable is immutable, checking its value at run time is waste of gas.

Recommendation Consider implementing two versions of the contract: one that tracks the total debt and another that don’t.

Client Comment *trackTotalDebt logic was fully removed from codebase.*

```
240 if (trackTotalDebt) {
```

```
354 if (trackTotalDebt) {
```

```
478 if (trackTotalDebt) {
```

```
955 if (trackTotalDebt) {
```

CVF-184. FIXED

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Recommendation This should be calculated in lazy way, i.e. only when the “quoteTokens-MaskInverted” value is actually used for the first time.

Client Comment *The mask is now computed only when required*

```
604 uint256 quotedTokensMaskInverted =
        supportsQuotas ? ~ICreditManagerV3(creditManager).
            ↪ quotedTokensMask() : type(uint256).max;
```

CVF-185. INFO

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Description This condition is checked twice.

Recommendation Consider checking once and reusing.

Client Comment *This would make the code less readable for very marginal gas savings.*

```
945 if (action == ManageDebtAction.INCREASE_DEBT) {
```

```
968 if (action == ManageDebtAction.INCREASE_DEBT) {
```

CVF-186. INFO

- **Category** Suboptimal
- **Source** CreditFacadeV3.sol

Description The value calculated here doesn't depend on the provided "callData".

Recommendation Consider not updating it here.

Client Comment *Currently, the computation consists of a possibly warmed up storage value multiplied by a constant. Pre-caching this value in storage would always necessarily do a cold read. Computing this dynamically, but outside the function, would make the code less coherent.*

```
988 maxQuota: uint96(Math.min(type(uint96).max, maxQuotaMultiplier *
    ↪ debtLimits.maxDebt))
```

CVF-187. FIXED

- **Category** Documentation
- **Source** CreditManagerV3_USDT.sol

Description This comment is misleading, as it doesn't describe the difference between this contract and "CreditManagerV3".

Recommendation Consider elaborating more.

Client Comment *Add a more fleshed out comment.*

```
9 /// @title Credit Manager
```

CVF-188. INFO

- **Category** Procedural
- **Source** CreditAccountV3.sol

Description We didn't review this file.

```
8 import {SafeERC20} from "@1inch/solidity-utils/contracts/libraries/
    ↪ SafeERC20.sol";
```

CVF-189. INFO

- **Category** Bad naming
- **Source** CreditAccountV3.sol

Description The function name seems unrelated to its functionality, as the function is actually very generic and allows to perform arbitrary external calls.

Recommendation Consider renaming or making the function more specific.

Client Comment *The NatSpec explains the purpose of the function.*

```
86 function rescue(address target, bytes calldata data)
```

CVF-190. INFO

- **Category** Procedural
- **Source** CreditConfiguratorV3.sol

Description We didn't review this file.

```
36 import {IPriceOracleV2} from "@gearbox-protocol/core-v2/contracts/
    ↪ interfaces/IPriceOracleV2.sol";
```

CVF-191. FIXED

- **Category** Procedural
- **Source** CreditConfiguratorV3.sol

Description While most of the modifiers are applied to “forbidToken”, the “nonUnderlying-TokenOnly” modifier is applied to “_forbidToken”. This makes the code more error-prone and harder to read.

Recommendation Consider applying all modifiers in the same place.

Client Comment *nonUnderlyingTokenOnly was moved to external function*.

```
338 function forbidToken(address token)
        external
340     override
        nonZeroAddress(token)
        pausableAdminsOnly // I:[CC-2A]
```

```
348 function _forbidToken(address _creditFacade, address token) internal
    ↪  nonUnderlyingTokenOnly(token) {
```

CVF-192. FIXED

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Recommendation Should be “prevCreditFacade”.

Client Comment *Change to a correct name.*

```
660 CreditFacadeV3 prevCreditFacace = CreditFacadeV3(creditFacade());
```

CVF-193. FIXED

- **Category** Bad datatype
- **Source** CreditManagerV3.sol

Recommendation The default value should be a named constant.

Client Comment *The value is now a named constant.*

```
95 uint8 public override maxEnabledTokens = 12;
```

CVF-194. FIXED

- **Category** Documentation
- **Source** CreditManagerV3.sol

Description The number format for these variables is unclear.

Recommendation Consider documenting.

Client Comment *Added a comment clarifying the format.*

98
```
uint16 internal ltUnderlying;
```

101
```
uint16 internal feeInterest;
```

104
```
uint16 internal feeLiquidation;
```

109
```
uint16 internal liquidationDiscount;
```

115
```
uint16 internal feeLiquidationExpired;
```

119
```
uint16 internal liquidationDiscountExpired;
```

CVF-195. FIXED

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Description This code looks weird.

Recommendation Consider either require all pools not to revert inside the “supportQuotas” function, or passing the “supportQuotas” flag as a constructor argument.

Client Comment *Was removed along with supportsQuotas.*

203
```
try IPoolV3(_pool).supportsQuotas() returns (bool sq) {
    supportsQuotas = sq; // I:[CMQ-1]
} catch {}
```

CVF-196. FIXED

- **Category** Bad datatype
- **Source** CreditManagerV3.sol

Recommendation The value “address(1)” should be a named constant.

Client Comment *The value is now a named constant.*

```
222 _activeCreditAccount = address(1);
```

```
656 if (creditAccount == address(1)) revert
    ↪ ActiveCreditAccountNotSetException();
```

CVF-197. FIXED

- **Category** Documentation
- **Source** CreditManagerV3.sol

Description The return value is not documented.

Recommendation Consider documenting.

Client Comment *Return value description added.*

```
242 returns (address creditAccount)
```

CVF-198. FIXED

- **Category** Procedural
- **Source** CreditManagerV3.sol

Description This comment is confusing, as in the structure, these fields go in different order.

Recommendation Consider listing the fields in their actual order.

Client Comment *Order rearranged.*

```
251 // newCreditAccountInfo.since = uint64(block.number); // U:[CM-6]
    // newCreditAccountInfo.flags = 0; // U:[CM-6]
    // newCreditAccountInfo.borrower = onBehalfOf; // U:[CM-6]
```

CVF-199. FIXED

- **Category** Documentation
- **Source** CreditManagerV3.sol

Description This argument is not documented.

Recommendation Consider documenting.

Client Comment *Added documentation for the parameter*.

```
305 CollateralDebtData calldata collateralDebtData,
```

CVF-200. FIXED

- **Category** Documentation
- **Source** CreditManagerV3.sol

Description The returned values are not documented.

Recommendation Consider documenting.

Client Comment *Added documentation for the values*.

```
315 returns (uint256 remainingFunds, uint256 loss)
```

CVF-201. FIXED

- **Category** Documentation
- **Source** CreditManagerV3.sol

Description The comment is inaccurate at this code zeros out not only the “borrower” field, but also fields “flags” and “since”.

Recommendation Consider making the comment more accurate.

Client Comment *Improved the comment*

```
326 // delete creditAccountInfo[creditAccount].borrower; // U:[CM-8]
```

```
328     let slot := add(currentCreditAccountInfo.slot, 4)
```

CVF-202. FIXED

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Description The condition “closureAction == ClosureAction.LIQUIDATE_ACCOUNT” is checked twice.

Recommendation Consider refactoring to check is once.

Client Comment *The function is stack bottlenecked, so it’s not possible to move this into a separate variable.*

352
```
liquidationDiscount: closureAction == ClosureAction.
    ↪ LIQUIDATE_ACCOUNT
```

355
```
feeLiquidation: closureAction == ClosureAction.LIQUIDATE_ACCOUNT ?
    ↪ feeLiquidation : feeLiquidationExpired,
```

CVF-203. FIXED

- **Category** Procedural
- **Source** CreditManagerV3.sol

Description The value “amountToPool + remainingFunds + 1” is calculated twice.

Recommendation Consider calculating once and reusing.

Client Comment *See above.*

364
```
if (underlyingBalance < amountToPool + remainingFunds + 1) {
```

369
```
            amount: _amountWithFee(amountToPool + remainingFunds + 1
                ↪  - underlyingBalance)
```

CVF-204. FIXED

- **Category** Procedural
- **Source** CreditManagerV3.sol

Recommendation Brackets are redundant.

Client Comment *supportsQuotas-related conditions were removed.*

491
```
uint128 quotaFees = (supportsQuotas) ? currentCreditAccountInfo.
    ↪ quotaFees : 0;
```

CVF-205. FIXED

- **Category** Documentation
- **Source** CreditManagerV3.sol

Description The returned value is not documented.

Recommendation Consider documenting.

Client Comment *Docstring added.*

```
541 returns (uint256 tokenMask)
```

CVF-206. FIXED

- **Category** Documentation
- **Source** CreditManagerV3.sol

Description The number format for these “uint16” arguments is unclear.

Recommendation Consider documenting.

Client Comment *Added clarifying comments to docstrings.*

```
673     uint16 minHealthFactor
```

```
714 function isLiquidatable(address creditAccount, uint16
    ↪ minHealthFactor) external view override returns (bool) {
```

```
788     uint16 minHealthFactor,
```

```
1457    uint16 _feeInterest,
        uint16 _feeLiquidation,
        uint16 _liquidationDiscount,
1460    uint16 _feeLiquidationExpired,
        uint16 _liquidationDiscountExpired
```

```
1484    uint16 ltInitial,
        uint16 ltFinal,
```

CVF-207. FIXED

- **Category** Documentation
- **Source** CreditManagerV3.sol

Recommendation Should be “externally” rather than “internally”.

Client Comment *Changed per recommendation.*

762
```
///      prevented from being called internally
```

CVF-208. FIXED

- **Category** Documentation
- **Source** CreditManagerV3.sol

Description This argument is not documented.

Recommendation Consider documenting.

Client Comment *Added a docstring.*

912
```
uint256[] memory collateralHints,
```

CVF-209. FIXED

- **Category** Bad datatype
- **Source** CreditManagerV3.sol

Recommendation The underlying token mask should be a named constant.

Client Comment *Replaced with a named constant.*

1190
```
tokenMask = (token == underlying) ? 1 : tokenMasksMapInternal[token
    ↪ ]; // U:[CM-34]
```

1226
```
if (tokenMask == 1) {
```

CVF-210. FIXED

- **Category** Unclear behavior
- **Source** CreditManagerV3.sol

Description This check should be performed only when token != underlying.

Client Comment *Refactored the function.*

```
1191 if (tokenMask == 0) revert TokenNotAllowedException(); // U:[CM-34]
```

CVF-211. FIXED

- **Category** Documentation
- **Source** CreditManagerV3.sol

Description The number format for these “uint16” returned values is unclear.

Recommendation Consider documenting.

Client Comment *Formats clarified in docstrings.*

```
1202 function liquidationThresholds(address token) public view override
       ↪ returns (uint16 lt) {
```

```
1213     returns (address token, uint16 liquidationThreshold)
```

```
1223     returns (address token, uint16 liquidationThreshold)
```

```
1282         uint16 _feeInterest,
             uint16 _feeLiquidation,
             uint16 _liquidationDiscount,
             uint16 _feeLiquidationExpired,
             uint16 _liquidationDiscountExpired
```

CVF-212. INFO

- **Category** Bad datatype
- **Source** AddressProviderV3.sol

Recommendation The hardcoded version numbers should be named constants.

Client Comment *In this particular case, we believe that writing the version number explicitly is more readable.*

```
65  return getAddressOrRevert(AP_CONTRACTS_REGISTER, 1); // F:[AP-4]
70  return getAddressOrRevert(AP_PRICE_ORACLE, 2); // F:[AP-5]
80  return getAddressOrRevert(AP_DATA_COMPRESSOR, 2); // F:[AP-7]
100 return getAddressOrRevert(AP_WETH_GATEWAY, 1); // F:[AP-11]
105 return getAddressOrRevert(AP_ROUTER, 1); // T:[AP-7]
```

CVF-213. FIXED

- **Category** Documentation
- **Source** AddressProviderV3.sol

Description The comment is exactly the same as for the “getWethToken” function.

Recommendation Consider rephrasing the comment to reflect difference with that function.

Client Comment *Comment changed.*

```
98  /// @return Address of WETH token
    function getWETHGateway() external view returns (address) {
```

CVF-214. FIXED

- **Category** Documentation
- **Source** BotListV3.sol

Description The returned value is not documented.

Recommendation Consider documenting.

Client Comment *Added a docstring.*

```
99 returns (uint256 activeBotsRemaining)
```

CVF-215. FIXED

- **Category** Suboptimal
- **Source** BotListV3.sol

Recommendation Consider refactoring as: if (permissions != 0) { if (forbiddenBot[bot]) revert(); ... }

Client Comment *Changed per recommendation.*

```
105 if (forbiddenBot[bot] && permissions != 0) {
        revert InvalidBotException(); // F: [BL-03]
    }
```

```
109 if (permissions != 0) {
```

CVF-216. FIXED

- **Category** Procedural
- **Source** BotListV3.sol

Description The expression “activeBots[creditAccount] is calculated twice.

Recommendation Consider calculating once and reusing.

Client Comment *Replaced with a single storage pointer computation.*

```
110     activeBots[creditAccount].add(bot); // F: [BL-03]
```

```
132 activeBotsRemaining = activeBots[creditAccount].length(); // F: [BL
    ↪ -03]
```

CVF-217. FIXED

- **Category** Suboptimal
- **Source** BotListV3.sol

Description Transferring fee on each transaction is inefficient.

Recommendation Consider collecting fee at the contract's balance, and transferring later in bulk.

Client Comment *The fee is now accumulated into a uint64 variable and transferred on overflowing uint64.max. The particular size is used to fit all non-immutables into a single slot.*

```
191 IERC20(weth).safeTransfer(treasury, feeAmount); // F: [BL-05]
```

CVF-218. FIXED

- **Category** Unclear behavior
- **Source** BotListV3.sol

Description These events are emitted even if nothing actually changed.

Client Comment *Added checks that values are changed.*

```
240 emit SetBotForbiddenStatus(bot, status);
```

```
252 emit SetBotDAOFee(newFee); // F: [BL-02]
```

CVF-219. FIXED

- **Category** Suboptimal
- **Source** AccountFactoryV3.sol

Description This function should emit some event.

Client Comment *Event added.*

```
135 function rescue(address creditAccount, address target, bytes
    ↪ calldata data)
```

CVF-220. FIXED

- **Category** Suboptimal
- **Source** BalancesLogic.sol

Recommendation This could be simplified as: forbiddenTokensOnAccount ^= tokenMask;

Client Comment *Simplified.*

```
114 +forbiddenTokensOnAccount &= forbiddenTokensOnAccount - 1;
```

CVF-221. FIXED

- **Category** Procedural
- **Source** BalancesLogic.sol

Description The expression ”forbiddenBalances[i]” is calculated several times.

Recommendation Consider calculating once and reusing.

Client Comment *Changed to ”forbiddenBalances[i] = BalanceWithMask(...);”.*

```
117 +forbiddenBalances[i].token = token; // U:[BLL-3]
    +forbiddenBalances[i].tokenMask = tokenMask; // U:[BLL-3]
    +forbiddenBalances[i].balance = IERC20(token).safeBalanceOf({account
       ↪ : creditAccount}); // U:[BLL-3]
```

CVF-222. INFO

- **Category** Readability
- **Source** ICreditFacadeV3Multicall.sol

Description Defining a top-level constant in a file named after an interface makes is harder to navigate through code.

Recommendation Consider either moving the constant definitions into the interface or moving them into a separate file.

```
43 +uint256 constant FORBIDDEN_TOKENS_BEFORE_CALLS = 1 << 192;
```

```
50 +uint256 constant REVERT_ON_FORBIDDEN_TOKENS_AFTER_CALLS = 1 << 193;
```

```
68 +uint256 constant EXTERNAL_CONTRACT_WAS_CALLED = 1 << 194;
```

```
77 +uint256 constant PAY_BOT_CAN_BE_CALLED = 1 << 195;
```

CVF-223. FIXED

- **Category** Documentation
- **Source** IBotListV3.sol

Description The format of this field is unclear.

Recommendation Consider documenting.

Client Comment *"BotSpecialStatus" is now removed, corresponding fields are documented in "BotInfo" struct.*

```
19 +uint192 specialPermissions;
```

CVF-224. INFO

- **Category** Bad datatype
- **Source** ControllerTimelockV3.sol

Recommendation The arguments types should be "IPoolV3" and "ICreditManagerV3" respectively.

```
351 +function getCreditManagerDebtLimit(address pool, address
    ↪ creditManager) public view returns (uint256) {
```

CVF-225. INFO

- **Category** Bad datatype
- **Source** ControllerTimelockV3.sol

Recommendation The arguments types should be "ICreditManagerV3" and "IERC20" respectively.

```
405 +function getLTRampParamsHash(address creditManager, address token)
    ↪ public view returns (uint256) {
```

CVF-226. FIXED

- **Category** Suboptimal
- **Source** ControllerTimelockV3.sol

Description Returning a hash as ”uint256” is unusual and confusing.

Recommendation Consider returning as ”bytes32”.

Client Comment *Fixed, although it still should be casted to ”uint256” later.*

405
```
+function getLTRampParamsHash(address creditManager, address token)
   ↪ public view returns (uint256) {
```

CVF-227. INFO

- **Category** Bad datatype
- **Source** ControllerTimelockV3.sol

Recommendation The arguments types should be ”IPoolQuotaKeeperV3” and ”IERC20” respectively.

471
```
+function getTokenLimit(address poolQuotaKeeper, address token)
   ↪ public view returns (uint96) {
```

507
```
+function getTokenQuotaIncreaseFee(address poolQuotaKeeper, address
   ↪ token) public view returns (uint16) {
```

CVF-228. INFO

- **Category** Bad datatype
- **Source** ControllerTimelockV3.sol

Recommendation The arguments types should be ”IGaugeV3” and ”IERC20” respectively.

609
```
+function getMinQuotaRate(address gauge, address token) public view
   ↪ returns (uint16) {
```

650
```
+function getMaxQuotaRate(address gauge, address token) public view
   ↪ returns (uint16) {
```

CVF-229. FIXED

- **Category** Bad naming
- **Source** PolicyManagerV3.sol

Description The semantics of the ”bool” returned value is unclear.

Recommendation Consider giving a descriptive name to the returned value and/or documenting.

Client Comment *Added comment.*

```
280 +function calcDiff(uint256 a, uint256 b) internal pure returns (
        ↪ uint256, bool) {
```

CVF-230. FIXED

- **Category** Suboptimal
- **Source** CreditConfiguratorV3.sol

Recommendation This could be simplified as: forbiddenTokensMask ^= mask;

Client Comment *Simplified.*

```
935 +forbiddenTokensMask &= forbiddenTokensMask - 1;
```

CVF-231. INFO

- **Category** Readability
- **Source** CreditFacadeV3.sol

Recommendation Consider including the mask of forbidden tokens into the error.

Client Comment *Acknowledged, though we now have more informative exceptions.*

```
1176 revert ForbiddenTokensException(); // U:[FA-27]
```

CVF-234. INFO

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Recommendation While Solidity doesn't support immutable variables of type "string", it is possible to pack/unpack a short string into bytes32 value very efficiently, and store the packed value in an immutable variable: https://gist.github.com/3sG-gpQ8H/567354534170905e047b299286697e19

```
220 +string public override name;
```

CVF-235. FIXED

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Recommendation Bitwise "or" would be more efficient, as it doesn't perform overflow checks.

Client Comment *This check is no longer needed (debt is no longer repaid on closure, and it is guaranteed to be non-zero on liquidation).*

```
515 +if (collateralDebtData.debt + profit + loss != 0) {
```

CVF-236. FIXED

- **Category** Suboptimal
- **Source** CreditManagerV3.sol

Recommendation This could be simplified as: tokensToTransferMask ^= tokenMask;

Client Comment *Simplified.*

```
2138 +tokensToTransferMask &= tokensToTransferMask - 1;
```

CVF-237. FIXED

- **Category** Suboptimal
- **Source** AccountFactoryV3.sol

Recommendation Usually mapping is used in similar cases. Array could be a bit more efficient, but limits the key size.

Client Comment *Replaced with a mapping.*

```
57 +mapping(address => QueuedAccount[2 ** 32]) internal _queuedAccounts
      ↪ ;
```

CVF-238. FIXED

- **Category** Suboptimal
- **Source** BotListV3.sol

Recommendation It would be more efficient to merge these mappings into a single mapping whose keys are credit managers and values are structs encapsulating the values of the original mappings.

Client Comment *Since bot's forbidden status is now global, we merged the mappings into one with bots as keys.*

```
86 +mapping(address => mapping(address => mapping(address => uint192)))
      ↪  public override botPermissions;
```

```
89 +mapping(address => mapping(address => mapping(address => BotFunding
      ↪ ))) public override botFunding;
```

```
94 +mapping(address => mapping(address => EnumerableSet.AddressSet))
      ↪ internal activeBots;
```

```
101 +mapping(address => mapping(address => BotSpecialStatus)) public
      ↪ override botSpecialStatus;
```

CVF-239. FIXED

- **Category** Procedural
- **Source** BotListV3.sol

Recommendation The expression ”botSpecialStatus[creditManager][bot] is calculated twice. consider calculating once and reusing.

```
161 +botSpecialStatus[creditManager][bot].forbidden
    +    || botSpecialStatus[creditManager][bot].specialPermissions != 0
```

CVF-240. FIXED

- **Category** Readability
- **Source** BotListV3.sol

Recommendation It would be more readable to make the loop counter go down from len-1 to zero.

```
220  for (uint256 i = 0; i < len; ++i) {
```

```
223 +    address bot = accountBots.at(len - i - 1); // U:[BL-6]
```

CVF-241. FIXED

- **Category** Procedural
- **Source** BotListV3.sol

Description The value ”balanceOf[account]” is read from the storage twice.

Recommendation Consider reading once and reusing.

Client Comment *This code no longer exists.*

```
500 +if (balanceOf[account] < amount) {
```

```
505 +    balanceOf[account] -= amount; // U:[BL-4,5]
```

CVF-242. INFO

- **Category** Suboptimal
- **Source** BalancesLogic.sol

Description Declaring a top-level enum in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the enum declaration into the contract or moving it into a separate file.

```
25 +enum Comparison {
```

CVF-243. INFO

- **Category** Bad datatype
- **Source** ControllerTimelockV3.sol

Recommendation The types for the arguments should be ”ICreditManagerV3” and ”IERC20” respectively.

```
297 +function getLTRampParamsHash(address creditManager, address token)
      ↪ public view returns (bytes32) {
```

CVF-244. INFO

- **Category** Documentation
- **Source** GearStakingV3.sol

Description These arguments are not properly documented with ”@param” tags.

Recommendation Consider documenting properly.

```
86 +uint8 v,
   +bytes32 r,
   +bytes32 s
```

CVF-245. INFO

- **Category** Procedural
- **Source** CreditFacadeV3.sol

Description Defining top-level constants in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the constant definitions into the contract or moving them into a separate file.

```
51 +uint256 constant OPEN_CREDIT_ACCOUNT_FLAGS =
   +    ALL_PERMISSIONS & ~(DECREASE_DEBT_PERMISSION |
      ↪ WITHDRAW_COLLATERAL_PERMISSION);
```

```
55 +uint256 constant CLOSE_CREDIT_ACCOUNT_FLAGS = ALL_PERMISSIONS & ~
      ↪ INCREASE_DEBT_PERMISSION;
```

```
57 +uint256 constant LIQUIDATE_CREDIT_ACCOUNT_FLAGS =
   +    EXTERNAL_CALLS_PERMISSION | ADD_COLLATERAL_PERMISSION |
      ↪ WITHDRAW_COLLATERAL_PERMISSION;
```

CVF-246. INFO

- **Category** Procedural
- **Source** CreditManagerV3.sol

Description The "isExpired" flag is checked twice.

Recommendation Consider refactoring to check once.

Client Comment *Almost all changes to this function result in the "stack too deep" error, so had to keep it like this.*

```
344 +liquidationDiscount: isExpired ? liquidationDiscountExpired :
       ↪ liquidationDiscount,
    +feeLiquidation: isExpired ? feeLiquidationExpired : feeLiquidation,
```

CVF-247. INFO

- **Category** Bad datatype
- **Source** BotListV3.sol

Recommendation The type of the first key should be ”ICreditManagerV3”.

```
93 +/// @dev Mapping credit manager => credit account => set of bots
      ↪ with non-zero permissions
   +mapping(address => mapping(address => EnumerableSet.AddressSet))
      ↪ internal _activeBots;
```

CVF-248. INFO

- **Category** Overflow/Underflow
- **Source** PriceOracleV3.sol

Description Phantom overflow is possible here.

Recommendation Consider using the ”mulDiv” function.

Client Comment *As with normal ”convertToUSD”, we’d prefer it to revert here because numbers can’t be that big for overflow to happen unless something goes terribly wrong.*

```
102 +return amount * price / scale; // U:[PO-11]
```

CVF-249. INFO

- **Category** Unclear behavior
- **Source** PriceOracleV3.sol

Description This logic shouldn’t be executed in case ”priceFeed” is zero.

Client Comment *True, though it doesn’t cost too much gas so we’d prefer to keep it as is at this stage.*

```
203  stalenessPeriod := shr(160, data)
     skipCheck := and(shr(192, data), 0x01)
     decimals := shr(200, data)
     useReserve := and(shr(208, data), 0x01)
    +trusted := and(shr(216, data), 0x01)
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting