GEARBOX BOTS & INTEGRATIONS SECURITY AUDIT REPORT

April 24, 2025

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

The Gearbox project enables leveraged trading and integrates with leading DeFi projects. This audit is specifically focused on the partial liquidation bot, designed to facilitate the partial liquidation of credited accounts. Additionally, the audit covers certain specific DeFi integrations, namely Convex, Curve, and Camelot.

1.4 Project Dashboard

Project Summary

Title	Description
Client	GearBox Protocol
Project name	v.3.1 Bots & Integrations
Timeline	07.03.2024 - 01.04.2025
Number of Auditors	3

Project Log

Date	Commit Hash	Note
07.03.2024	ff9e3317f41d31413bc5436bd64e4f13b2a78f0e	initial commit for the audit (bot)
07.03.2024	2575396b2c933953483dd85cb2d5900134349f80	initial commit for the audit (integrations)
14.03.2024	405ec3afab3b9bedce77f5f0faa9979dfe2acb41	update of the code (bot)
27.03.2024	8e30305e526e2420b99bf6add36784d759faaf29	additional functionality and fixes (integration)
27.03.2024	dc7fe47f5b0c05d24f8349ed41bdd72f4989bf40	commit with fixes (bot)
02.04.2024	c4a8bda931db1982d8768b3ed7e75b9a10ed1189	fixed M.5 (integrations)
24.07.2024	e2fcd9aef437a19a37fdbd95cde2b7df16bac346	commit for the diff audit (integrations)
24.07.2024	807ccc7b7a9366522339e499ef1a14d20c27dbde	commit for the diff audit (bot)
09.08.2024	d427a3a5623f10bec704a8bfaaa1b587a1a30499	commit with the diff audit fixes (integrations)

Date	Commit Hash	Note
12.12.2024	799c30c366f0c03950c47b25fb32972449e94f58	commit with EqualizerRouterAdapter (v3.0)
12.12.2024	02474b8bdbff11b8ee24b0f04cf1408dcf46fc3d	commit with EqualizerRouterAdapter (v3.1)
19.03.2025	3d56f6ccfc202e52487ec9651babdd4fe5cb5788	commit for the diff audit (bot)
01.04.2025	9e56cb66d59ab27bad0c04339d3b401c230e7ae2	commit with minor changes (integrations v3.1)

Project Scope

The audit covered the following files:

File name	Link
contracts/bots/PartialLiquidationBotV3.sol	PartialLiquidationBotV3.sol
contracts/adapters/camelot/CamelotV3Adapter.sol	CamelotV3Adapter.sol
contracts/adapters/convex/ConvexV1_BaseRewardPool.sol	ConvexV1_BaseRewardPool.sol
contracts/adapters/curve/CurveV1_StableNG.sol	CurveV1_StableNG.sol
contracts/adapters/AbstractAdapter.sol	AbstractAdapter.sol
contracts/adapters/curve/CurveV1_Base.sol	CurveV1_Base.sol
contracts/adapters/velodrome/VelodromeV2RouterAdapter.sol	VelodromeV2RouterAdapter.sol
contracts/adapters/equalizer/EqualizerRouterAdapter.sol (v3.1)	EqualizerRouterAdapter.sol
contracts/adapters/equalizer/EqualizerRouterAdapter.sol (v3.0)	EqualizerRouterAdapter.sol

Deployments

Deployment verification will be conducted via https://permissionless.gearbox.foundation/bytecode/.

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	5
Low	5

ID	Name	Severity	Status
M-1	Unsafe support of non-standard ERC-20 tokens	Medium	Fixed
M-2	Incorrect conditions of enabling collateral tokens	Medium	Acknowledged
M-3	Maximum 2 instead of 12 extra reward tokens are supported	Medium	Fixed
M-4	Adding/replacing a new reward token is not supported	Medium	Acknowledged
M-5	Lack of path validation in the `Velodrome` adapter	Medium	Fixed
L-1	Unvalidated user input in the `_getAddLiquidityOneCoinCallData` function	Low	Acknowledged
L-2	An unvalidated index in some helper functions	Low	Acknowledged
L-3	Lack of validation for the `creditAccount` parameter	Low	Acknowledged
L-4	Error handling syntax improvements	Low	Acknowledged
L-5	Outdated year in file headers	Low	Fixed

1.6 Conclusion

This audit focused on several adapter contracts and a bot contract responsible for executing partial liquidations:

- **CamelotV3Adapter:** Integration adapter for the Camelot AMM project.
- **ConvexV1BaseRewardPoolAdapter:** Integration adapter for the `BaseRewardPool` contract of the Convex.finance infrastructure.
- **CurveV1AdapterStableNG:** Integration adapter for Curve Stableswap-ng pools.
- **VelodromeV2RouterAdapter:** Integration adapter for Velodrome AMM project (added during the second audit iteration).
- **PartialLiquidationBotV3:** Partial liquidation bot contract.

Additionally, a review of out-of-scope contracts integrated into the adapters and a high-level review of the core codebase repository were conducted.

Partial Liquidation Bot

The partial liquidation bot introduces an option for liquidators to perform partial liquidations on accounts with a health factor below a safe threshold, which is especially useful in scenarios with low market liquidity. It offers a lower discount compared to full liquidation, making it less attractive for general use. Optionally, a bot with "soft liquidation" mode can be deployed via a specific configuration. This mode triggers partial liquidations before reaching the full liquidation threshold, aiming to gradually improve the account's health factor.

Attack Vectors and Concerns

- **Dealing with a sharp decrease in health factor:** Liquidation executions consume a significant amount of gas. Combined with potentially rising gas prices and oracle lag during volatile periods, transaction costs might outweigh potential rewards for liquidators, rendering them unprofitable for regular users. Gearbox's internal mechanisms might handle such situations, potentially absorbing losses with treasury funds to prevent bad debt.
- **Limited collateral seizure:** Currently, partial liquidations can only seize a single collateral token. This limitation hinders the efficient liquidation of accounts with diversified collateral positions.
- **Reentrancy:** The bot's use of a hardcoded call sequence to the CreditFacade nonreentrant multicall function combined with the usage of whitelisted collateral and underlying tokens effectively prevents reentrancy through the bot itself.

Adapter Integrations

The adapter audit focused on verifying proper integration with corresponding contracts and the implementation of comprehensive validation checks. These checks ensure adapter interactions only occur with whitelisted tokens and contracts, preventing unintended behavior.

Attack Vectors and Concerns:

- **Camelot Adapter:** Validates swap paths for multi-pool swaps to ensure they only involve whitelisted pools. Additionally, `tokenIn` and `tokenOut` must be recognized and whitelisted by the CreditManager system. The overall integration is considered safe.
- **Convex Adapter:** Validates reward and staked tokens during deployment, ensuring they are recognized and whitelisted by the system. Only the first two external reward tokens are automatically enabled. Additional tokens require explicit enabling through CreditFacade multicall execution.
- **Curve Adapter:** Limited to pools with a maximum of four tokens, despite the existence of Curve Stableswap-ng pools that can hold up to eight tokens within a single contract. All tokens within the pool, including the liquidity provider token, must be recognized and validated during deployment.
- **Velodrome Adapter:** While whitelisting ensures allowed pools and tokens for swaps, there's a potential concern for transferring tokens out of the system, bypassing the full collateral check on withdrawals.
- **Calldata Integrity:** All adapters were examined to verify the integrity of calldata passed for execution via calls within the CreditAccount contract. The conformity of arguments passed from adapters to integrated contracts was thoroughly checked.
- **Reentrancy:** All adapters strictly validate the set of tokens and addresses used during external contract execution, mitigating reentrancy risks.

The detailed issues and areas for improvement have been documented and presented in the detailed findings section of this report.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Unsafe support of non-standard ERC-20 tokens
Severity	Medium
Status	Fixed in dc7fe47f

Description

When retrieving the underlying tokens from the liquidator, the code does not perform checks for the return value. Although standard implementations of ERC-20 will revert on a failed PartialLiquidationBotV3.sol#L202, some implementations (e.g., USDT) may just return false instead. This could be passed without generating the appropriate exception. Consequently, no tokens would actually be retrieved from the liquidator, yet the collateral might still be withdrawn to the liquidator's account.
This issue is rated as MEDIUM severity because it is unlikely that it could be exploited in the current code base, given the other checks in place after liquidation. However, future development of the code could potentially increase the impact of this issue.

Recommendation

We recommend using `safeTransferFrom` instead to ensure that various ERC-20 implementations are correctly handled.

Client's commentary

> Fixed in PR-4.

M-2	Incorrect conditions of enabling collateral tokens
Severity	Medium
Status	Acknowledged

Description

In favor of gas optimization, the current implementation aims to maintain a token amount equal to 1 instead of 0. According to this rule, a token amount of 1 should be interpreted as having zero value, and consequently, the use of this token as collateral should not be enabled. However, the current implementation incorrectly handles corner cases where the token amount is incremented multiple times - the token may remain in a disabled state even when the amount is greater than 1.

This issue is rated as MEDIUM severity because it may cause unexpected behavior in smart contracts in corner cases. However, it is unlikely that it can be exploited in the current code base.

Related code:

- add_liquidity in Curve: CurveV1_StableNG.sol#L39-L40
- remove_liquidity_imbalance in Curve integration: CurveV1_StableNG.sol#L92-L94

Recommendation

We recommend refining the conditions for enabling tokens as collateral, considering the corner case described above.

Client's commentary

> Fully handling this edge case would require balance checks on all inbound tokens in the adapter, which would add a significant gas overhead. This unexpected behavior can only be realized due to very specific user input that is highly unlikely to ever occur under normal usage, and cannot be used to exploit the contracts. Hence, we do not believe that fixing this justifies additional gas overhead.

M-3	Maximum 2 instead of 12 extra reward tokens are supported
Severity	Medium
Status	Fixed in 501b6cee

Description

The Convex ExtraRewardStashV3 contract allows the owner to add up to 12 tokens, while the Gearbox adapter supports only 2. As a result, the adapter cannot be used for pools with more than 2 reward tokens.

This issue is rated as MEDIUM severity because pools with 3 or more reward tokens will require redeployment of the adapter contract, but the likelihood of this issue is low.

Related code: only 2 reward tokens are supported - ConvexV1_BaseRewardPool.sol#L80-L91

Recommendation

We recommend implementing support for the same number of reward tokens as in the Convex pool.

Client's commentary

> We prefer to use immutables (for gas optimization) to store data that changes never, or very rarely, such as extra reward token addresses in this case. The original 2-token limit was kept to avoid contract code bloat, as 2 tokens is enough to handle an overwhelming majority of pools.
> We have extended this to 4 tokens, as some pools on Arbitrum Aura have more than 2 due to ARB also being distributed as an extra reward. We do not see any reason to increase the number of supported tokens further.
> Fix: 501b6cee

M-4	Adding/replacing a new reward token is not supported
Severity	Medium
Status	Acknowledged

Description

The Convex pool allows the owner to add new reward tokens at any time and also replace one reward token with another. If such an event occurs, the new token will not be added to the Gearbox protocol and will remain in the balance of the credit account even if it is closed. Consequently, new users may receive accounts with foreign rewards.

This issue is rated as MEDIUM severity because it can be resolved by timely redeployment of the adapter contract, but the likelihood of this issue is low.

Related code: reward tokens are initialized only in the constructor - ConvexV1_BaseRewardPool.sol#L80-L91

Recommendation

We recommend tracking changes in the Convex protocol, performing timely redeployment of the adapter contract, or including logic for adding reward tokens.

Client's commentary

> Additional logic to add and replace reward tokens would make the contract more cumbersome and also require `rewardTokensMask` variable to be non-immutable, which will increase the gas overhead on withdrawals. We believe that redeploying the contract to handle the changed extra reward list is more efficient.
> Gearbox is able to handle unknown tokens appearing on a Credit Account, as one can simply transfer any token to it. As such, any new rewards on Convex side will not produce any issues and will become claimable by users after adapter redeployment and adding them as collateral.

M-5	Lack of path validation in the `Velodrome` adapter
Severity	Medium
Status	Fixed in c4a8bda9

Description

The swap functions `swapExactTokensForTokens` and `swapDiffTokensForTokens` of `VelodromeV2RouterAdapter` accepts `routes` parameter with arbitrary consequences of routes, consisting of (`tokenIn`, `tokenOut`, `stable`, `factory`) parameters. Neither `Adapter` or `Router` performs a check that the `tokenOut` parameter of the previous route matches the `tokenIn` parameter of the successive route. This may cause unexpected bahaviour, i.e. seizing the tokens by LP of `Velodrome` pools, which is unintended.

Recommendation

We recommend improving the validation of the `routes` parameter in the `Velodrome` adapter.

Client's commentary

> Fixed in c4a8bda9.

2.4 Low

L-1	Unvalidated user input in the `_getAddLiquidityOneCoinCallData` function
Severity	Low
Status	Acknowledged

Description

The `_getAddLiquidityOneCoinCallData` function utilizes variable `i`, which is an arbitrary value passed by the user, without any validation from the `add_liquidity_one_coin` function. If this value is outside the valid range, the transaction will be reverted due to built-in range checks.

Related code: add_liquidity_one_coin - CurveV1_Base.sol#L325-L333

Recommendation

We recommend adding an assertion to explicitly validate any values passed by the user.

Client's commentary

> Validation of `i` was present in Curve adapters previously but was later removed as redundant, since the validity of coin / underlying coin under `i` is checked both on the Curve contract side and Credit Manager side (since it validates all input and output tokens to be valid collateral). Lack of explicit validation can only lead to non-verbose errors due to incorrect data being passed by the user, so we do not see a compelling reason to add more checks.

L-2	An unvalidated index in some helper functions
Severity	Low
Status	Acknowledged

Description

The following function expects value [0..3] as input, but does not implement asserting for the unexpected value, potentially provided by the user:

- _get_token
- _get_underlying
- _get_token_mask
- _get_underlying_mask
- _approveTokens

Related code: the functions enlisted above CurveV1_Base.sol#L559-L597

This may lead to unexpected behavior.

Recommendation

We recommend adding an assertion to explicitly validate any values passed by the user.

Client's commentary

> Validation of `i` was present in Curve adapters previously but was later removed as redundant, since the validity of coin / underlying coin under `i` is checked both on the Curve contract side and Credit Manager side (since it validates all input and output tokens to be valid collateral). Lack of explicit validation can only lead to non-verbose errors due to incorrect data being passed by the user, so we do not see a compelling reason to add more checks.

L-3	Lack of validation for the `creditAccount` parameter
Severity	Low
Status	Acknowledged

Description

The issue is identified in the PartialLiquidationBotV3.sol#L124 function of the `PartialLiquidationBotV3` contract.

The `partiallyLiquidate` function accepts an arbitrary `creditAccount` value, which may not be a valid credit account. If a malicious contract address is provided instead of a valid credit account address, the caller of the `partiallyLiquidate` function could lose any assets they have approved for spending by the `PartialLiquidationBotV3` contract.

This issue is difficult to exploit because the `creditAccount` value is provided by the caller of the `partiallyLiquidate` function, making them responsible for validating the value to avoid their own losses. Failure to validate the value will not lead to losses for anyone else. However, in some rare scenarios, the caller may be confused and may suffer a loss of their assets.

Recommendation

We recommend ensuring that the `creditAccount` value is a valid credit account served by the credit manager and registered in the system.

Client's commentary

> After upgrading to v3.1, we no longer have a single contracts discovery entry-point like `contractsRegister` was, so ensuring that `creditAccount` is valid might be rather cumbersome and expensive (which we clearly want to avoid here). It's also safe to assume that liquidators are capable of (and should be responsible for) preventing loss of funds.

L-4	Error handling syntax improvements
Severity	Low
Status	Acknowledged

Description

The code uses `if (!statement) revert error;` for error handling, which is less readable than using the `require(statement, error);` syntax introduced in Solidity version 0.8.26. This impacts the clarity and maintainability of the code.

The issue is classified as low severity because it does not affect the functionality but rather the readability and standardization of the code.

Recommendation

We recommend updating the Solidity version to 0.8.26 and rewriting the error handling statements using the `require` function for better readability and adherence to Solidity best practices.

Client's commentary

> Aware of that, but we try to avoid using the most recent compiler versions, so only upgraded to a relatively mature 0.8.23.

L-5	Outdated year in file headers
Severity	Low
Status	Fixed in d427a3a5

Description

The issue is identified within the files of the integrations-v3 repository.

The file headers include a copyright notice with the year "2023". This year may become outdated if the contract is used or updated in subsequent years. An outdated year in the file header can lead to confusion regarding the contract's maintenance and versioning.

The issue is classified as low severity because it is related to copyright and does not impact the contract's functionality.

Recommendation

We recommend updating the year in the file headers to reflect the current year, "2024", to ensure accuracy and clarity in the contract's maintenance and versioning.

Client's commentary

> Fixed in d427a3a5.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes