PUBLIC

Code Assessment

# of the Migration Bot Smart Contracts

July 25, 2025

Produced for

Gearbox

by

CHAINSECURITY

Contents

1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 8
4 Terminology 9
5 Open Findings 10
6 Resolved Findings 11
7 Notes 12

1 Executive Summary

Dear Gearbox Team,

Thank you for trusting us to help Gearbox with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Migration Bot, according to Scope to support you in forming an opinion on their security risks.

Gearbox implements the migration bot, a module that facilitates the migration of liquidity by closing a credit account and opening another one.

The most critical subjects covered in our audit are the functional correctness of the implementation, the possible execution flows, and the safety of the funds. Security regarding all the aforementioned subjects is high.

The general subjects covered are gas efficiency, authorization, documentation, and testing. As the Gearbox protocol interacts with many different assets and protocols, thorough e2e testing of the migration of accounts with different configurations and assets would benefit the implementation. Currently, the testing is quite limited. Security regarding the rest of the aforementioned subjects is high.

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

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1
• Code Corrected	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Migration Bot repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	14 July 2025	f349923539cfb2136c0599b1f16e16b7fa7892fd	Initial Version
2	24 July 2025	f30752716e83ac3f8c6e7b7b97b57ac95e2a8c76	Fixes
3	25 July 2025	0b1827d061f1d5340b71e6670bd3c56b47f55070	Release 3.1

For the solidity smart contracts, the compiler version `0.8.23` was chosen.

The following contracts are considered in scope:

`contracts/migration/`:

- `AccountMigratorAdapter.sol`
- `AccountMigratorAdapter30.sol`
- `AccountMigratorAdapter31.sol`
- `AccountMigratorBot.sol`

2.1.1 Excluded from scope

All contracts not explicitly mentioned in scope are considered out of scope. In particular, the `AccountMigratorPreviewer.sol` is out of scope. The code is assumed to be deployed on chains that are equivalent to the Ethereum EVM. Vectors that include users purposely signing transactions that lead them to a loss of funds were not considered. The contracts in scope are assumed to be parametrised and deployed correctly.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Gearbox offers an implementation of the Migrator Bot. The purpose of the bot is to facilitate migration of a credit account (source CA) to new one (target CA). There are very minimal assumptions on what the involved credit managers (CM) of these accounts should be; they can be different versions or use a different pool. For the source CA to transfer its assets to the bot, an external call to an adapter is used (AccountMigratorAdapter).

Migration Flow:

1. The owner of the source CA calls `migrateCreditAccount` on the bot providing the needed parameters.
2. The parameters are validated.
3. The price feeds of the instance are updated.
4. The bot initiates a `botMulticall` on the source CA. For that the owner of the CA must have given the appropriate permissions.
5. The multicall eventually interacts with `MigratorAdapter` which makes the source CA call `migrate` on the bot itself.
6. The bot withdraws the collateral from the source credit account.
7. The bot opens the target CA by calling the Credit Facade of the new Credit Manager.
8. The target CA is configured (collateral is added, quotas are set, debt is increased).
9. The target CA optionally withdraws some of its collateral in underlying to the source CA.
10. The control returns back to source CA which finishes its closure e.g., by decreasing its debt to 0, appropriately upgrade the quotas, etc.

It is important to note the following:

- The source CA might not be empty after the migration but it shouldn't have any debt. Its owner can then withdraw its remaining assets.
- The migrator bot can be configured to properly interact with the phantom tokens.
- In principal, a migration could manually be performed without the owner of the CA. The migration bot however, enables the user to borrow with its target CA to repay the debt of the old source. This is achieved by interleaving the closure and opening operations i.e., the new credit account is opened while source credit account is in the process of closing. The health check at the end of the operations guarantees that both the source and the target accounts are healthy.

Migration Constraints:

Here we present some of the constraints of the migration

- the CMs should be known to the gearbox instance.
- both the source and target CMs should have all the migrated tokens enabled.
- the collateral to be migrated (can be phantom tokens).
- swap and other external calls on the opening shouldn't interact with a migrator adapter.

Handling Phantom Tokens

Phantom tokens are non transferable tokens used for accounting within the system when an actual token is not available. During the migration the phantom tokens of the source account must be withdrawn. The admins of the bot can override the default call (`withdrawPhantomToken`) when needed. Note that such functionality is not needed when Phantom Tokens are deposited to the target CA. The bot can handle the case where the phantom token position on the source and target CAs is handled by different contracts.

2.3 Trust Model

We infer the following trust model for the system:

- The owner of the bot is **fully trusted** as it configures how the phantom tokens are handled.
- All the components the system interacts with are assumed to function properly.

- The owner of the source credit account is assumed to provide the correct parameters for the migration.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies

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

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

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

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

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	1

- Usage of ptOverride Code Corrected

Informational Findings	1

- Missing Lock Check Code Corrected

6.1 Usage of `ptOverride`

Design Low Version 1 Code Corrected

CS-GEAR-PERIPH-002

`ptOverride` seems to need to always be populated for all phantom tokens. If it isn't, the underlying must be `0`. This will result in the comparison between the `uniqueTransferred` tokens and the `migratedCollateral.underlying` to fail. However, later `_getPhantomTokenWithdrawalCall` seems to be able to handle the case where `ptOverride` is empty for a phantom token.

Code corrected:

The `_validateParameters` function has been modified to correctly handle tokens which don't have a `ptOverride`. It now only checks that the `underlying` of the collateral matches the `ptOverride` if we have an override.

6.2 Missing Lock Check

Informational Version 1 Code Corrected

CS-GEAR-PERIPH-001

In order for a credit account to interact with the `AccountMigratorAdapter`, it needs to be unlocked by the `AccountMigratorBot`. The current implementation of the locking mechanism never checks the state of the lock before changing. `locked` is set to `true` without checking that the lock is unlocked. In case of an unexpected execution trace, such a check could prevent the migration from malfunctioning.

Code corrected:

The implementation now includes a check to ensure the lock is in the expected state before attempting to set or release it. If the lock is found to be in an incorrect state, the transaction reverts.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Migration With Phantom Tokens Deposit

Note **Version 1**

The migration of a credit account might fail should it hold phantom tokens. These cases might include, but are not limited to, the following:

- The phantom token implements a time delay. In this case, the withdrawal of a phantom token might fail if more time is required.
- The phantom token doesn't implement deposit. In this case, even if the token is successfully withdrawn from the source credit account, its underlying won't be able to be converted into a phantom token in the target account.

7.2 No Check on Balances After Account Closure

Note **Version 1**

When an account is being closed, a specific vector of calls is executed. However, there's no check regarding the state of the account's balances after the closing operation besides the health check enforced by the protocol. Such a check could allow users to ensure that only the expected amount has been transferred and the rest remains in the source CA to be withdrawn.

The same holds for the target CA. However, the migration flow allows the target CA to execute arbitrary opening calls, which could include checking the balances.

Moreover, note that the parameters of the migration are determined by the `AccountMigratorPreviewer` (out-of-scope). However, the previewer doesn't create a call to check the expected balances on the accounts after the migration.