PUBLIC

Code Assessment

of the Gearbox V3 Governance Smart Contracts

December 8, 2023

Produced for

by

Contents

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 Gearbox V3 Governance according to Scope to support you in forming an opinion on their security risks.

Gearbox implements the governance module for Gearbox V3.

The most critical subjects covered in our audit are the functional correctness of the contracts, their configuration, and the interaction with the rest of the Gearbox system. Only minor issues were uncovered which have been addressed. Security regarding all the aforementioned subjects is high.

The general subjects covered are access control, documentation and specification, gas efficiency, and the complexity of the implementation. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

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 Gearbox V3 Governance 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	11 Nov 2023	414df8ebf45ad6d09de3a24426e214300177e5f2	Initial Version
2	27 Nov 2023	6637d9df56b09615a9e88370d90a4dadcb4cd7dd	Fixes
3	06 Dec 2023	c90434702c163f3f1c2cb4db90cece525160ee07	Check for EOAs

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

In scope are the following contracts:

- Create2Factory.sol
- GovernorV3.sol
- interfaces/IGovernorV3.sol
- interfaces/ITimelock.sol

2.1.1 Excluded from scope

Any contracts not explicitly listed above are out of the scope of this review. Third-party libraries such as OpenZepplin's or contracts with which the GovernorV3 interacts such as the Timelock are also out of the scope of this review.

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 implements a governance module for Gearbox V3. Readers should refer to the relevant report from ChainSecurity to get a full overview of the protocol.

The Governance module consists of two smart contracts. The governor and the factory.

2.3 Factory

This contract implements a factory that makes use of deterministic deployments (CREATE2). It allows a user to deploy a smart contract by calling `deploy` and configure it by calling `callExternal[WithValue]` setting the newly deployed contract as the `target`.

2.4 GovernorV3

This contract is used for the management of the Gearbox protocol. The Governor interacts with a Timelock contract similar to one implemented by Uniswap . The timelock contract maintains a queue of transactions that can be executed after some time. A transaction is defined by a target contract, the ETH to be attached to the call, the signature of the function to be called, the parameters of the call, and the time after which it can be executed. The Governor defines two distinct roles, the queue admin which is the DAO's multisig, and the veto admin, a multisig with a lower signature threshold than the DAO's multisig. The queue admin can:

- queue a single transaction by calling `queueTransaction()`,
- queue a batch of transactions by first calling `startBatch()` and then repeatidly `queueTransaction()` while in the same block. These calls are all batched in one transaction. It is assumed that there's enough gas space to both create and execute the batch.

The veto admin can:

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

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

2.5 Trust Model and Roles

We extracted the following trust model from the codebase:

- The queue admin i.e., the DAO multisig: It inherits the trust assumptions for the GearboxDAO and its members. Its role is described in detail in the section above.
- The veto admin: It is considered trusted. It should not be able to harm the system but it can cause disturbance as it can block its configuration.
- The owner of the factory is considered trusted.

2.6 Changes in Version 2

In version 2 the following changes were implemented:

- Batched transactions can be added only by the admin, who started the batch so that a malicious admin cannot add a transaction to the batch. Note that a malicious admin could in theory observe the mempool and front-run any other actions from another admin by starting a new batch. This would cause all subsequent actions added in that block by an admin other than the malicious one to fail. The issue can be circumvented with the use of private mempools.
- The *eta* parameter for batched transactions is defined on a per-batch basis.

2.7 Changes in Version 3

In version 3 the following changes were implemented:

- The Governor contract can be set to reject non-EOA users who try to call `executeTransaction()` or `executeBatch()`. This can happen by setting the `isExecutionByContractsAllowed` variable. The modification of the variable can only be invoked by the timelock contract. It is important to highlight, that limiting the calls to EOAs

limits composability as no contracts can execute these transactions and potentially bundle them with other transactions.

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 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 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

- Missing Input Sanitization Code Corrected

6.1 Missing Input Sanitization

Design Low Version 1 Code Corrected

CS-GEARGOV-002

1. The addresses in the constructor of `GovernorV3` are not sanitized.
2. The addresses given to `GovernorV3.addQueueAdmin()` and `GovernorV3.updateVetoAdmin()` are not sanitized. Even though the veto admins are expected to prevent setting addresses wrong, the contract logic should also prevent it.
3. The `eta` parameter of the `GovernorV3.queueTransaction` is only sanitized by the `Timelock` contract as to whether the execution time is beyond some minimum required delay. Moreover, an action can be executed within a time window. However, there's no check on whether all the batched actions can be executed within the same window. Thus, a batch could be submitted but not be able to be executed.

Code corrected:

In version 2, the *eta* is defined on a per-batch basis. All transactions in the batch should have the same *eta*.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Veto Admin Can Veto Its Own Update

Informational **Version 1**

CS-GEARGOV-001

In the `GovernorV3` (`Governor` in V2) contract, with the assumption that `vetoAdmin` is a multisig requiring fewer signatures than the `queueAdmins`, it is theoretically easier to compromise `vetoAdmin` rather one of the `queueAdmin`. If some keys of the `vetoAdmin` multisig are compromised, all the queued transactions could be vetoed including a veto admin update. This issue is only relevant if the multisig does not allow the rest of the signers to replace the compromised keys.