PUBLIC

Code Assessment

of the Permissionless Smart Contracts

Aug 01, 2025

Produced for

Gearbox

by

CHAINSECURITY

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 Permissionless according to Scope to support you in forming an opinion on their security risks.

Gearbox implements a new governance system which aims to enable different risk curators to run their own gearbox markets. The new system allows the migration of the legacy system into the new system.

The most critical subjects covered in our audit are the correct instantiation of all system components, the migration logic of the legacy system, and the upgradeability, configurability, and liveness of the system. In the current implementation, neither the current system (Shutting down a market configurator) nor the legacy system can be fully configured (Legacy CreditManager cannot be fully configured). The migration of the legacy system is underspecified as it's not known which components of the legacy system will immediately be upgraded to newer versions. Moreover, the liveness of the system can be harmed in some cases (Reverting proposals lock cross-chain governance). Finally, upgrading some components of the system is not possible (Factory migration will fail).

The general subjects covered are functional correctness, gas consumption, testing, and documentation and specification. Testing was very limited in the first iteration of the report. This led to a substantial number of functional correctness issues (Timelock transactions can be executed before the ETA) that could have been prevented. Testing was significantly improved in subsequent versions. Some of the operations executed by Governance have very high gas requirements. Documentation is sufficient. However, some parts are underspecified (Signatures On Different Chains).

In summary, we find that the system 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	7
• Code Corrected	7
Medium-Severity Findings	9
• Code Corrected	9
Low-Severity Findings	9
• Code Corrected	6
• No Response	3

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 Permissionless 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	19 Jan 2025	6c1b6ac842b751198e82db67ce5beb4a1c079d68	Initial Version
2	27 Feb 2025	5a9a1d337f3a4f4a0cde66674f30f138f7a6ca90	First batch of fixes
3	28 Feb 2025	dad69fb90f0805907359b247b80ebb7ed5245112	Second batch of fixes
4	11 Apr 2025	6ef5000e1b642f3476946595777fb841c81af0f0	Final Version
5	14 Jul 2025	68f4a0d3812a110a5cbc0c567519740362ae78b2	RateKeeper Patch
6	25 Jul 2025	4f678ac1ee24372ad6f472e3e0b3b7db2fe1e658	RateKeeper Patch V2

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

The following contracts are in scope:

```
factories:
    AbstractFactory.sol
    AbstractMarketFactory.sol
    CreditFactory.sol
    InterestRateModelFactory.sol
    LossPolicyFactory.sol
    PoolFactory.sol
    PriceOracleFactory.sol
    RateKeeperFactory.sol
global:
    BytecodeRepository.sol
    CrossChainMultisig.sol
helpers:
    DefaultIRM.sol
    DefaultLossPolicy.sol
    EIP712Mainnet.sol
    ProxyCall.sol
instance:
    AddressProvider.sol
    InstanceManager.sol
    MarketConfiguratorFactory.sol
    PriceFeedStore.sol
libraries:
    CallBuilder.sol
    ContractLiterals.sol
    Domain.sol
    NestedPriceFeeds.sol
market:
    ACL.sol
    ContractsRegister.sol
    Governor.sol
    MarketConfigurator.sol
    TimeLock.sol
    TreasurySplitter.sol
    legacy:
        MarketConfiguratorLegacy.sol
traits:
    DeployerTrait.sol
    ImmutableOwnableTrait.sol
```

During the assessment the following commits from Core were considered:

- V3.0 (legacy): b959c4b642ca5099d037f53464be1269071216d8
- V3.10: 66f9b8e7be833964c935cd43d1e8002b9f08d9ea before Version 2
- V3.10: 562ccc19210fe43c170c4451eb35fb786982ca43 after Version 2

After Version 2, the following contracts were added to the scope:

```
helpers:
    DefaultDegenNFT.sol
```

2.1.1 Excluded from scope

Any contracts not explicitly listed above are out of the scope of this review. Third-party libraries are out of the scope of this review, especially `LibString` and `SSTORE2` are assumed to work correctly. We assume all the chains where Gearbox is going to be deployed on preserve the semantics of EVM on mainnet.

3 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 a governance and deployment system for the Gearbox Protocol across EVM chains. The system comprises the following building blocks:

- cross-chain governance
- bytecode security and versioning
- instance management
- market deployment and management

- legacy system integration

3.1 Cross-chain governance

Through CrossChainMultisig, the cross-chain governance module is responsible for managing the governance of the Gearbox Protocol across different EVM chains. The goal of the module is to guarantee a consistent state across all the different chains. This is achieved by executing the same proposals on all chains in the same order. The correct ordering is ensured by a system of hash-chain.

On Ethereum Mainnet, Gearbox DAO will vote on proposals that can ultimately be submitted to the CrossChainMultisig. Proposals must be submitted along with the hash of the previously executed proposal. Then, a set of trusted signers has to sign the proposal with the previous hash, when the threshold is reached, the proposal is executed. A proposal is a set of calls with a target address and a target chain. It is possible to target all the chains by setting the target chain to 0. Moreover, a proposal must target all chains if its target is CrossChainMultisig. This guarantees that the state of CrossChainMultisig is synced on all chains.

On chains other than Mainnet, the signed proposals that were executed on Mainnet (reached the threshold) can be permissionlessly submitted to the CrossChainMultisig. The proposal will be executed if the hash of the proposal matches the hash of the last executed proposal on the target chain and the target chain id matches as well.

Note that multiple proposals targeting the same prevProposalHash can coexist, but only the first one to receive enough signatures will be executed and the others will be invalidated since the prevProposalHash will be updated.

Below is a non-exhaustive list of cross-chain governance actions:

- add and remove signers, and modify the signatures threshold on the CrossChainMultisig
- deploy system contracts on a chain, see Instance management for more details
- manage auditors and domains on the BytecodeRepository
- add an already deployed market to a market configurator through the crossChainGovernanceProxy
- finalize the migration of a legacy market and configure its linked GEAR staking contract

Note that governance actions in general can be calls to arbitrary targets with arbitrary calldata.

3.2 Bytecode security and versioning

The BytecodeRepository contract is deployed along the instance and is owned by the crossChainGovernanceProxy, which can ultimately be called by the CrossChainMultisig. The BytecodeRepository holds the bytecode of almost all contracts to be used within the Gearbox ecosystem. Each contract has a type and a version. The contract type is built as follows: DOMAIN::POSTFIX, some domains can be marked as public domains by the governance. The version is built as follows: XYZ where X is the major version, Y is the minor version, and Z is the patch version.

Contract developers can permissionlessly submit their bytecode to the repository, then trusted auditors can post a signature containing the bytecode hash and the URL of their report. To be allowed to be deployed, the bytecode must be audited by at least one auditor and must be either marked as a system contract by the governance or be in the public domain.

The deployment of an approved contract is permissionless but is expected to be performed through trusted callpaths (InstanceManager, MarketConfigurator, ...) for contracts used by the Gearbox Protocol and be properly parametrized.

3.3 Instance management

There will be at most one instance of Gearbox Protocol per chain and each instance will be controlled by a trusted instance manager. The instance manager is the owner of the `InstanceManager` contract and is responsible for managing the allowed price feeds in the `PriceFeedStore`. The `InstanceManager` contract itself is the entry point in the system for the governance, instance manager, and treasury, as it controls the three proxies `crossChainGovernanceProxy`, `instanceManagerProxy`, `treasuryProxy`.

The `InstanceManager` is expected to be deployed with the governance as the primary owner. Upon deployment of the `InstanceManager`, the three aforementioned proxies are deployed, as well as the `BytecodeRepository` and the `AddressProvider` contracts. The instance can then be activated and the ownership is transferred to the instance manager.

The governance can deploy or redeploy some system contracts through the `InstanceManager` that will be recorded in the `AddressProvider`. Such contracts are:

- Bytecode repository
- Cross-chain governance
- The three proxies mentioned above
- Instance Manager
- Treasury
- GEAR token
- Price Feed Store
- Factory contracts (`MarketConfiguratorFactory`, `PoolFactory`, `PriceOracleFactory`, `InterestRateModelFactory`, `RateKeeperFactory`, `LossPolicyFactory`)
- GEAR staking contract
- Bot list

3.4 Market deployment and management

Within a Gearbox instance, a market curator can create various different markets via a MarketConfigurator contract. The contract specifies an admin which is expected to be a Timelock controlled by a Governor contract (see Gearbox Governance) even though this is not strictly enforced. A market configurator is deployed together with an ACL (Access Control List), a contract register, and a treasury splitter.

Markets: The lending pool is the epicenter of a market. Even though the components of a market are thoroughly discussed in the security review of the Core we explain them here briefly. A market consists of:

- A pool that is always deployed with a quota keeper i.e., the contract that controls the quota for the lending pool.
- A price oracle i.e., the entry point for price queries.
- An interest rate model which calculates the interest rate based on the pool's utilization.
- A ratekeeper contract which determines the cost of maintaining some quota for a token.
- A loss policy that determines the behavior of the market during liquidations with bad debt.

The pool, the price oracle, and the loss policy are registered in the contract register. All the components except for the pool itself are updatable. Upon updating, a hook is invoked notifying the rest of the components.

A market can be shut down as long as there's no debt to be repaid to the pool. When a market is shut down, the ratekeeper is frozen if it's a Gauge i.e., a contract that determines the quota rates based on the votes of users.

Configuration: The `MarketConfigurator` has the configurator role for all the contracts of the market. The contracts are deployed through factories. When the market configurator wants to make a call to a market component it relies on the factory of this component to construct the calldata for the call. This requires the factory to be authorized for this component (see migration). If an action must be applied on multiple components, then it is executed via a market hook which is a call aiming at all factories of the market. All actions of the contract are controlled by the admin (usually a timelock) with the exception of the emergency configuration which is invoked by the emergency admin.

Credit Suite: A market might be associated to some credit suites. A credit suite consists of an account factory, a credit manager, a credit configurator, and a credit facade. A credit suite can be shut down as long as it has no outstanding debt for the pool.

Migration: Whenever a new patch is available for a factory, the market can migrate to use it. As multiple components might have been authorized for a factory (e.g., the pool factory is authorized for both the pool and the quota keeper), all the components need to authorize the new factory.

Upgradeability: In general the following components can be changed for the market:

- Rate Keeper
- Interest Rate Model
- Price Oracle
- Loss Policy
- Credit Facade
- Credit Configurator

On the other hand, the Credit Account Factory, the Credit Manager and the Pool and the Quota Keeper cannot be upgraded.

3.5 Legacy system migration

The market configurator system cannot be directly used for the legacy system i.e., the Gearbox markets already deployed. In order to make the legacy system compatible with the new governance architecture, `MarketConfiguratorLegacy` is introduced which extends the `MarketConfigurator` contract. The contract does the following:

- It wraps the deployment of the legacy system into a market configurator i.e., creates a market without deploying a new contract by simply authorizing the appropriate factories and targets.
- It creates an ACL which will co-exist with the already deployed legacy ACL for the contracts, as the legacy contracts do not allow the ACL to be reset. The legacy market configurator becomes the owner of the legacy ACL as well as a pausable and an unpausable admin.

3.6 Trust Model and Roles

- governance signers: trusted to sign only proposals coming from the DAO on Mainnet. If enough signers collude off-chain, they can sign and execute a proposal that was not voted on Mainnet, thus leading a non-mainnet chain to a different chain and forbidding any further updates.
- instance manager: the instance manager is the owner of the `InstanceManager` after activation. They are responsible for whitelisting the price feeds of the instance they supervise in the `PriceFeedStore`. They are trusted to act in the best interest of the protocol and the users. By whitelisting a bad price feed, they can increase the risk of liquidations or loss for the system.

- bytecode submitters: not trusted as anyone can submit bytecode. Submitting broken or adversarial bytecode should not hurt the system as auditors must sign the bytecode before deployment.
- auditors: trusted to guarantee the security and compatibility of the bytecode they sign with the rest of the system. If a buggy, insecure, or incompatible bytecode is signed, the system can be at risk of loss of funds and/or liveness if the bytecode is deployed. Moreover, auditors are assumed to sign code that preserves the semantics of the interface that the code intends to implement.
- treasury: it can access the assets collected for the Gearbox treasury and distribute fees collected by a market configuration. The role doesn't have to be fully trusted as it can only propose changes in the Treasury splitter which require the consent of the risk curator.
- risk curators: they manage a `MarketConfigurator` through the `Governor/Timelock` or directly. They are trusted to manage market parameters in the best interest of the users. By choosing bad underlying tokens or other market parameters the risk curators can increase the risk of liquidations and bad debt in the markets they oversee.
- emergency admins: each `MarketConfigurator` has an emergency admin that is not constrained to a timelock. They can set some preset parameters to halt/pause parts of a market or credit suite. They are trusted to act in the best interest of the protocol and the users.

3.7 Version 2

Version 2 introduces the following changes:

- `CrossChainMultisig` implements recovery mode. It can be enabled if enough signatures have been collected. During recovery mode, all calls of a batch are skipped. In the fixes, the feature was modified to only skip calls that don't target the `CrossChainMultisig` and it can be enabled on a per-chain basis. The recovery mode can be disabled by calling `disableRecoveryMode()`.
- The handling of the system and public domains is improved. The system domains cannot have any owner.
- `DefaultDegenNFT` is introduced, which is a default implementation of the DegenNFT.

4 Limitations and use of report

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

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

5 Terminology

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

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

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

Likelihood	Impact		
	High	Medium	Low
High	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.

6 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
- Correctness: Mismatches between specification and implementation

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	3

- Duplicated Events
- Overwriting Addresses
- tokenSplits() Does Not Return Default Split

6.1 Duplicated Events

Design Low Version 1

CS-GEARGOV310-001

The following events are emitted twice:

- `GrantRole` in `MarketConfigurator` is also emitted by `ACL`
- `RevokeRole` in `MarketConfigurator` is also emitted by `ACL`

6.2 Overwriting Addresses

Design Low Version 1

CS-GEARGOV310-002

`AddressProvider.setAddress()` allows to overwrite the value of a key-version pair. This shouldn't be allowed as it cancels the effect of versioning i.e., maintaining all the different values of a key. This is not important as `setAddress()` is permissioned and the users who can modify the address provider are trusted by the system.

6.3 `tokenSplits()` Does Not Return Default Split

Design Low Version 1

CS-GEARGOV310-003

The `TreasurySplitter.tokenSplits()` returns the token splits for a given token, but the function will return a zero-split instead of the default split if the token split is not initialized.

7 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	7

- Cannot Forbid an Adapter With Normal Configuration Code Corrected
- Factory Migration Will Fail Code Corrected
- Legacy CreditManager Cannot Be Fully Configured Code Corrected
- Overwritten Version in AddressProvider Code Corrected
- Reverting Proposals Lock Cross-Chain Governance Code Corrected
- Timelock Transactions Can Be Executed Before the ETA Code Corrected
- Updating a Rate Keeper Will Freeze the Epoch Code Corrected

Medium-Severity Findings	9

- Batching disableRecoveryMode Can Be Problematic Code Corrected
- Recovery Mode Message Replay Code Corrected
- Dynamic Types Must Be Hashed for EIP712 Code Corrected
- Free Choice of maxEnabledTokens Can Be Dangerous Code Corrected
- Legacy PriceOracle Cannot Be Updated Code Corrected
- Missing Sanity Checks on minorVersion Code Corrected
- Rate Keeper Cannot Be Updated Code Corrected
- Shutting Down a Market Configurator Code Corrected
- Threshold Not Enforced When Removing Signer Code Corrected

Low-Severity Findings	6

- A Transaction Can Be Canceled After Execution Code Corrected
- Allowance Timestamp Can Be Reset Code Corrected
- Enough Admins Check Code Corrected
- Forbidden initCode Code Corrected
- Pending Owners Code Corrected
- Wrong Specifications Code Corrected

Informational Findings	5

- Dead Code Code Corrected
- Inconsistent Sanity Check for Versioning Code Corrected
- Redundant Chunk Code Corrected
- Unnecessary Imports Code Corrected

- Wrong Variable Name Code Corrected

7.1 Cannot Forbid an Adapter With Normal Configuration

Correctness High Version 1 Code Corrected

CS-GEARGOV310-009

In `CreditFactory.configure()`, in the callpath to forbid an adapter, the adapter is authorizing the factory again instead of removing the authorization. Since the adapter already authorized the factory, this will revert as calling `authorizeFactory()` on an already-authorized target will fail.

Code corrected:

The callpath has been corrected to unauthorize the factory.

7.2 Factory Migration Will Fail

Correctness High Version 1 Code Corrected

CS-GEARGOV310-010

When processing the targets in `MarketConfigurator._migrateFactoryTargets()`, the array of targets is trimmed by one element at every iteration, and the next index to read is incremented. This leads to an out-of-bound array access as soon as the `targets` array contains more than one element.

Code corrected:

The loop iterates over a copy of the array cached in memory, and the elements are removed from the array in storage.

7.3 Legacy CreditManager Cannot Be Fully Configured

Design High Version 1 Code Corrected

CS-GEARGOV310-016

The legacy `CreditManager` exposes two functions `setMaxEnabledTokens()` and `makeTokenQuoted()`, callable by the `CreditConfigurator`. But when migrated under the `MarketConfiguratorLegacy`, there will be no way to call those functions as `setMaxEnabledTokens()` and `makeTokenQuoted()` are not valid calls to be encoded from the `CreditFactory`. This will block further attempts to change the maximum number of enabled tokens as well as mark an already added token as quoted once the legacy credit suite is migrated.

Code corrected:

The new `core-v3` `CreditConfiguratorV3` implements a function to make all the tokens quoted on the legacy `CreditManagers` with version `< 3_10`. Gearbox wants the maximum number of enabled tokens to be immutable in `3_10` and thus does not expose a function to modify this number.

7.4 Overwritten Version in `AddressProvider`

Correctness High Version 1 Code Corrected

CS-GEARGOV310-011

The function `AddressProvider._setAddress()` computes the subversions from `version` instead of `_version`. This has the effect that the subversions will always end up with the same values (i.e., `300` and `310`) even if the targeted version is `320`, breaking the versioning system.

Code corrected:

The function has been refactored and merged into `setAddress()`.

7.5 Reverting Proposals Lock Cross-Chain Governance

Design High Version 1 Code Corrected

CS-GEARGOV310-017

The `CrossChainMultisig` uses `Address.functionCall()` to execute the proposals, which will bubble up the revert if the inner call fails and make the whole proposal execution fail. If a proposal that was executed on Mainnet reverts on another chain, the `CrossChainMultisig` of that other chain will not be able to execute any other Mainnet governance proposal, as it would have to execute the reverting proposal first.

This can end up in two distinct scenarios:

a. The cross-chain governance is locked forever

b. The trusted signers have to create a "fork" from the Mainnet proposals hash chain to be able to execute new proposals

---

Code corrected:

The `CrossChainMultisig` implements a recovery mechanism that can be triggered by a threshold of signers. While in recovery mode, the proposals are added to the proposals chain, but are not executed. The recovery mode can be disabled later. This system allows proposals to revert without locking the cross-chain governance or requiring a fork.

7.6 Timelock Transactions Can Be Executed Before the ETA

Correctness High Version 1 Code Corrected

CS-GEARGOV310-012

In the function `Timelock.executeTransaction()`, the block timestamp is enforced to be smaller or equal to the ETA instead of greater or equal. This allows a transaction to be executed only before the ETA, which contradicts the intended functionality of the timelock.

Code corrected:

The inequality has been reversed and the function reverts if the block timestamp is strictly smaller than the ETA.

7.7 Updating a Rate Keeper Will Freeze the Epoch

Correctness High Version 1 Code Corrected

CS-GEARGOV310-013

In `RateKeeperFactory.onUpdateRateKeeper()`, if the type of the new ratekeeper is a `GAUGE`, the epoch is unfrozen on the old ratekeeper instead of the new one. This will keep the epoch frozen and prevent the new ratekeeper from updating the rate. To fix the state, the factory must be updated with the possibility to unfreeze the epoch.

Code corrected:

The epoch is unfrozen on the new ratekeeper instead of the old one.

7.8 Batching `disableRecoveryMode` Can Be Problematic

Design Medium Version 2 Code Corrected

CS-GEARGOV310-027

The `disableRecoveryMode()` call can be batched with other transactions. However, if any transaction in the batch fails, the recovery mode cannot be used to recover. There's no constraint that enforces that `disableRecoveryMode()` cannot be batched with other transactions.

Code corrected:

If a batch contains a `disableRecoveryMode()` call, the `CrossChainGovernance` ensures this call to be the only transaction in the batch.

7.9 Recovery Mode Message Replay

Design Medium Version 2 Code Corrected

CS-GEARGOV310-030

A recovery mode message can be replayed across different chains, leading to unintended consequences. Consider the following scenario:

1. Chain A enters recovery mode at some point.
2. Later, Gearbox is deployed on Chain B.

3. Unless `executeBatch` is carefully constructed, a signature from Chain A can be reused to forcefully skip batches on Chain B until the next `disableRecoveryMode` is called.

This could allow malicious actors to manipulate governance execution, potentially causing inconsistencies across chains.

Code corrected:

The recovery mode message includes the target chain ID to mitigate cross-chain replay.

7.10 Dynamic Types Must Be Hashed for EIP712

Correctness Medium Version 1 Code Corrected

CS-GEARGOV310-014

Following EIP-712, dynamic types such as `string` and `bytes` must be encoded as the keccak256 hash of their content (https://eips.ethereum.org/EIPS/eip-712#definition-of-encodedata). In `CrossChainMultisig.hashProposal()`, `call.Data` should be hashed as its type is `bytes`.

Code corrected:

The dynamic types are correctly encoded EIP712 style.

7.11 Free Choice of `maxEnabledTokens` Can Be Dangerous

Design Medium Version 1 Code Corrected

CS-GEARGOV310-023

The free choice of the value for `maxEnabledTokens` can be problematic if the value is too big and the gas required to liquidate the account exceeds the block gas limit. This could lead to bad debt in the protocol.

Code corrected:

This issue was fixed by a code change in the `core-v3` repository. The `CreditManagerV3` enforces 20 as a maximum value for `maxEnabledTokens`. Our report for core-v3 covers this change.

7.12 Legacy `PriceOracle` Cannot Be Updated

Design Medium Version 1 Code Corrected

CS-GEARGOV310-024

During the update of the `PriceOracle`, the old price oracle is queried with `reservePriceFeeds()`, but the legacy `PriceOracleV3` does not implement the function, reverting the call and blocking the update.

Code corrected:

The function PriceOracleFactory._getPriceFeed() was updated to call priceFeedsRaw() on the price oracle with version lower than 3_10.

7.13 Missing Sanity Checks on minorVersion

Design Medium Version 1 Code Corrected

CS-GEARGOV310-032

When creating a new market or a new credit suite, the admin can use an arbitrary minor version for the factories. This means that the interfaces of the various components might not match. For example, an admin could deploy a pool using a factory with minor version 2x0 and a credit manager using version 3x0. These can happen from a market configurator with version 4x0. There's no guarantee that such a configuration can work as expected.

Coed corrected:

The MarketConfigurator enforces the version to be 3_XY.

7.14 Rate Keeper Cannot Be Updated

Correctness Medium Version 1 Code Corrected

CS-GEARGOV310-015

When updating the rate keeper through the market configurator, the onUpdateRateKeeper is triggered on the PoolFactory. The pool factory executes setGauge on the quota keeper. The quota keeper then checks that all the tokens have been added to the new Rate Keeper (the Gauge). However, the tokens are added during a later hook on the RateKeeperFactory. Therefore, the update will revert.

The issue was found by Gearbox during an independent review.

Code corrected: The tokens are now added as part of the install hook after the new rate keeper has been constructed.

7.15 Shutting Down a Market Configurator

Correctness Medium Version 1 Code Corrected

CS-GEARGOV310-031

When shutting down a market configurator, the configurator is added to the _shutdownMarketConfiguratorsSet by the following snippet.

```
if (_shutdownMarketConfiguratorsSet.add(marketConfigurator)) {
    revert MarketConfiguratorIsAlreadyShutdownException(marketConfigurator);
}
```

Note, however, that EnumerableSet.add returns true when an element is successfully added to the set. In this case, the revert will be executed, rendering a market shutdown impossible.

Code corrected:

The condition was updated to revert if the market configurator is already in the set.

7.16 Threshold Not Enforced When Removing Signer

Design Medium Version 1 Code Corrected

CS-GEARGOV310-025

When a signer is removed from `CrossChainMultisig`, the function does not enforce that enough signers are left to meet the threshold. If the threshold cannot be met, proposals cannot be executed anymore.

Code corrected:

The remaining number of signers after a removal is enforced to be at least the threshold.

7.17 A Transaction Can Be Canceled After Execution

Design Low Version 1 Code Corrected

CS-GEARGOV310-019

It is possible to call `Timelock.cancelTransaction()` on a transaction that has already been executed, which will emit the `CancelTransaction` event.

Code corrected:

The function returns early if the transaction is in the queue.

7.18 Allowance Timestamp Can Be Reset

Design Low Version 1 Code Corrected

CS-GEARGOV310-018

The function `PriceFeedStore.allowPriceFeed` does not revert if the price feed is already allowed for use with the token. This allows the function to be called again to reset the `allowanceTimsetamp` to the current block timestamp.

Code corrected:

The function reverts if the price feed is already allowed for use with the token.

7.19 Enough Admins Check

Design Low Version 1 Code Corrected

CS-GEARGOV310-022

In the `Governor`, if ownership was renounced when an admin (queue or execution) is removed, there is no check enforcing that at least one admin remains. There is also no check to enforce that at least one execution admin is set when permissionless execution is disabled. If all the admins are removed or if permissionless execution is disabled and no execution admin is set, the governor will be locked forever.

Code corrected:

The owner cannot renounce ownership of the contract. As the owner can also queue and execute transactions, this ensures the `Governor` will always have at least one admin.

7.20 Forbidden initCode

Design | Low | Version 1 | Code Corrected

CS-GEARGOV310-028

In `BytecodeRepository`, the `initCode` is checked to not be forbidden twice, once when submitted and another time when deployed. The check can easily be circumvented by adding some garbage byte when submitted.

Code corrected:

The init code is not checked to be forbidden upon submission.

7.21 Pending Owners

Design | Low | Version 1 | Code Corrected

CS-GEARGOV310-026

For `PriceFeedStore._validatePriceFeedDeployment()`, the pending owner of the priceFeed is checked. There seems to be circular dependency here: The PriceFeedStore can indeed accept the ownership of an `Ownable2Step` only if the price feed is known. However, the pricefeed cannot be added to the known ones if it doesn't get validated.

Code corrected:

`PriceFeedStore._validatePriceFeedDeployment()` now accepts the ownership of the price feed.

7.22 Wrong Specifications

Design | Low | Version 1 | Code Corrected

CS-GEARGOV310-020

1. The specs of `EIP712Mainnet._cachedDomainSeparator` variable mention the domain separator being recomputed if the chain ID doesn't match the cached one, but the domain separator is recomputed only if the `address(this)` does not match the cached one.

2. The specs of the `BytecodeRepository.contractTypeOwner` mapping has the comment "// if contractType is public", but the semantics of the mapping is to connect a contract type to its owner, regardless of the contract type being in the public domain
3. The specs of `CrossChainMultiSig.signProposal` mention that "// Executed by any signer to make cross-chain distribution possible". However, cross-chain distribution is not restricted by the execution of this function on mainnet. It is enough to collect the signatures off-chain to execute a proposal on another chain.

Code corrected:

1. The comment was removed
2. The mapping was removed
3. The specs were updated

7.23 Dead Code

Informational | Version 1 | Code Corrected

CS-GEARGOV310-006

Some parts of the code are unused. Unused code should be removed from the codebase to improve clarity and maintainability.

1. the `ProxyCallExecuted` event in `ProxyCall` is never used

Code corrected:

The event definition has been removed.

7.24 Inconsistent Sanity Check for Versioning

Informational | Version 1 | Code Corrected

CS-GEARGOV310-029

`BytecodeRepository._validateVersion()` checks that the version is between 100 and 999. However, `AddressProvider._validateVersion()` only checks that the version is above 100.

Code corrected:

The version validation is now consistent.

7.25 Redundant Chunk

Informational | Version 1 | Code Corrected

CS-GEARGOV310-007

`BytecodeRepository._writeInitCode()` splits the init code into chunks as follows:

```
uint256 len = initCode.length / chunkSize + 1;
```

This allows for the following theoretical issue: in case `initCode.length % chunkSize == 0`, the last chunk will be empty.

Code corrected:

The number of chunks is now calculated as follows. Therefore, no redundant chunk is created:

```
uint256 len = (initCode.length - 1) / chunkSize + 1;
```

7.26 Unnecessary Imports

Informational Version 1 Code Corrected

CS-GEARGOV310-008

Some of the imports inthe codebase are not used. Here is a non-exhaustive list:

- `IAddressProvider` in `Domain.sol`
- `AP_INSTANCE_MANAGER` in `Domain.sol`
- `AP_ADDRESS_PROVIDER` in `InstanceManager.sol`
- `AP_MARKET_CONFIGURATOR_FACTORY` in `InstanceManager.sol`
- `AP_INSTANCE_MANAGER_PROXY` in `CreditFactory.sol`
-

Code corrected: The redundant imports have been removed.

7.27 Wrong Variable Name

Informational Version 1 Code Corrected

CS-GEARGOV310-021

1. The variable `minorVersion` in `AddressProvider._setAddress()` represents the major version
2. The variable `patchVersion` in `AddressProvider._setAddress()` represents the minor version

Code corrected:

The function has been refactored and merged into `setAddress()`, which uses the correct variable names.

8 Informational

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

8.1 Gas Optimizations

Informational **Version 1**

CS-GEARGOV310-004

1. In `TreasurySplitter._distribute()`, the check `receiver != address(this)` is unnecessary as the function `_setSplit` does not allow a receiver to be the `TreasurySplitter` contract address.
2. In `CrossChainMultisig._verifySignatures()`, checking that `_signers` contains `signer` is redundant as if multiple proposals are competing, only one fo them will be executed, and the other discarded. There is no possibility for a proposal to be by a signer that will be removed before the proposal is executed.

8.2 Missing Events

Informational **Version 1**

CS-GEARGOV310-005

Smart contracts should emit events for all important state changes. The following states changes may deserve an event:

1. In `TresurySplitter`, when a new proposal is added
2. In `TresurySplitter`, when a new proposal is executed
3. In `TresurySplitter`, when a proposal is canceled

9 Notes

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

9.1 Gas Consumption of Market Configuration Operations

Note Version 1

The market configuration might include some gas-heavy operations such as hooks on different factories. The same applies to market configurator legacy. An example is the migration of the legacy system in the construction of the `MarketConfiguratorLegacy`. We assume that Gearbox will make sure that all operations fit in the block gas limit of each chain.

9.2 Signatures On Different Chains

Note Version 1

`BytecodeRepository` expects to make use of signatures submitted for mainnet on any chain. These signatures contain a domain separator which depends on the address of the `BytecodeRepository`. Therefore, it also depends on the address of the `InstanceManager` deploying it, on the current chain. If the contracts are not deployed on the same address as on mainnet then the signatures cannot be verified. Reproducible deployment is not always trivial.

Certain networks and deployment strategies may pose challenges to this requirement:

1. Optimism-stack-based chains: when bridging funds to an OP chain, the nonce of the `from` account is incremented. See https://specs.optimism.io/protocol/deposits.html#nonce-handling. This should be taken into account in cases where the deployer of the `InstanceManager` contract has bridged funds. This would increase their nonce and thus affect the address of the deploying contract as the address depends on the nonce of the deployer. Consequently, the address of the token contract will be also affected as it depends on the address of the CREATE2 caller (which is the deploying contract). Affected chains: Optimism, Base, etc.
2. Since the same EOA needs to be used for `InstanceManager` deployment on all the networks, the private key of the EOA must be used to sign multiple transactions. Factors such as bad setup, insider threat, malicious code or infrastructure, bad key generation, etc. may lead to the private key leak. It is important that the deployer contract does not have any special permissions and does not have access to funds.
3. `zkSync Era`: CREATE_PREFIX used in zkSync is different from the mainnet one and will produce a different address for the same bytecode. In addition, to use CREATE2 in `Deployer` contract, zkSync requires that the deployed contract bytecode is already known to the compiler as states here. Thus, the `BytecodeRepository.deploy()` function won't work on zkSync.

9.3 Transferability of Shares

Note Version 1

Users should be aware that on legacy pools, shares are always transferable, in comparison to the newer pools (>=3.10) where transfers are paused when the pool is paused.

9.4 CrossChainMultisig Must Enforce Identical Signers Across Chains

Note Version 1

When constructing CrossChainMultisig (CCM), it is crucial to ensure that the same signers are added across all chains. Otherwise, the following issue can arise:

1. Suppose signers on Chain A during construction are {S1, S2}, but on Chain B, it is only {S1}.
2. If a transaction is created to add S2 on Chain B, it will fail on Chain A because S2 is already present.
3. Similarly, removing a signer on one chain but not the others may create inconsistent governance states.