PUBLIC

Code Assessment

of the Chief Smart Contracts

March 21, 2025

Produced for

Sky

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

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

Sky implements a continuous approval voting system introducing SKY as the governance token while simplifying the previous version.

The most critical subjects covered in our audit are functional correctness, front-running, and suitability for Sky's governance. The general subjects covered are trustworthiness and gas efficiency.

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	0

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 Chief 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	17 Mar 2025	11f7c6d893ef13932bcb01cf27d6b2d6ea58f7df	Initial Version

For the solidity smart contracts, the compiler version `0.8.21` was chosen and the `evm_version` was set to `shanghai`.

The following contracts were in scope:

```
src/Chief.sol
```

2.1.1 Excluded from scope

Generally, all files not mentioned above are out of scope.

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.

Sky implements `Chief`, a governance contract for continuous approval voting, to replace the existing `MCD_ADM` `DSChief`. The change aims to set SKY as the governance token while removing unused code and adjusting the flashloan protection to reduce the occurrence of blocked withdrawals.

General. The `Chief` implements a continuous approval voting system, allowing voters to signal approval for up to `maxYays` candidates. Each candidate in such an ordered set, referred to as `slate`, receives an approval equal to the voting power which corresponds to the locked amount of governance tokens `gov`. More specifically, the mechanism allows any candidate to become the leader `hat` if it has a higher approval than the current leader. To effectively become the `hat`, an eligible candidate has to be lifted.

In the context of Sky governance, the `hat` receives privileges to call functions that are access-controlled by `Chief.canCall()`. Notable examples are the scheduling of spells in the governance timelock `DSPause` and the execution of emergency actions (e.g. through Mom contracts).

Voters can manage their power with the following functions:

- `lock()`: Locks the governance token and increases the voter's voting power by an equal amount.
- `free()`: Unlocks the governance token and reduces the voter's voting power by an equal amount. Note that the function cannot be called if `lift()` has previously been called in the same block.

The approval voting operations include the functions below:

- `etch()`: Publishes a `slate` (recall that this is an ordered set of approved candidates).
- `vote()`: Approves the candidates within the given `slate` and consequently removes approval for the prior slate. Note that a second version is implemented allowing to provide the set of candidates (`etch()` is executed first in this case).
- `lift()`: Replaces the `hat` with the candidate if the candidate's approval is greater. Note that two consecutive `lift()` operations in two distinct blocks must respect a cooldown period `liftCooldown`.

Disallowing calls to `free()` if `lift()` has been called before in the same block prevents flashloan-like scenarios from occurring within one block. The `liftCooldown` provides a non-blocking window for withdrawals.

Deployment and Launch. Upon contract deployment, the contract is not yet operative. Its `live` storage variable is set to `0` which causes `canCall()` to return `false`. Nonetheless, voters can acquire voting power and cast votes even while the contracts is not live which is required to make the contract operative. More specifically, the `launch()` function allows setting `live` to `1` if the 0-address is the `hat` (value has never changed, or it has been lifted) and if it satisfies the minimum approval `launchThreshold` for launching. The purpose of the launch procedure is to avoid granting rights to leaders which have low approval when the contract has just been deployed and voter participation is still low.

2.3 Trust Model

The only actors within the `Chief` are voters holding no privileges. Voters are untrusted. However, their actions are expected to not self-inflict significant financial harm.

The majority of voting power is typically trusted so that no malicious proposals are elected as the leader. However, it is expected that the governance timelock `MCD_PAUSE DSPause` has a timelock suitable for the `Chief` so that proposals can be dropped in time if necessary. Further, non-emergency actions are expected to be executed by the `MCD_PAUSE_PROXY DSPauseProxy` and that it is governed by `MCD_PAUSE`. Last, it is expected that emergency actions cannot create a profit for any address.

Further, note that the governance token `gov` is expected to be SKY and is fully trusted.

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 our findings. The findings are split into these different categories:

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 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.

6.1 Considerations Regarding Lift Cooldown

Note Version 1

The parameter `liftCooldown` restricts the frequency of calls `lift()` over several blocks to provide a window for non-blocking calls to `free()`. Governance should consider the below when selecting the `liftCooldown` value:

- `liftCooldown>=1` should hold. Otherwise, an attacker could `lift()` in every block to DoS the withdrawals.
- `liftCooldown` should be large enough to facilitate the withdrawals by users and VoteDelegate contracts.
- `liftCooldown` should be small enough to allow for lifting emergency spells that can drop malicious proposals.

6.2 Differences Between New And Old Chief Contracts

Note Version 1

The new Chief contract preserves the core functionality of the old Chief contract while introducing a number of changes to improve the efficiency and security. Users and integrators should be aware of the following differences:

Dependency

- DSAuth has been removed since the old Chief has no owner and is the authority of itself.
- DSRoles has been removed since it is never used.
- DSNote has been removed, and respective events are defined and emitted.

Interface

- The new Chief is expected to work with SKY instead of MKR, and the `GOV` interface has been replaced to `gov`.
- The IOU token has been removed.
- The `live` flag (used to be `bool`) has been changed to `uint256`, while its value can only be 0 or 1.
- Users' individual `last` tracker (updated upon `lock()`) has been replaced by a global `last` tracker (updated upon `launch()` and `lift()`).
- The storage variable `MAX_YAYS` has been replaced with an immutable `maxYays`.
- The constant `LAUNCH_THRESHOLD` has been replaced with an immutable `launchThreshold`.
- A constant `EMPTY_SLATE` has been added.
- An immutable `liftCooldown` has been added.

Spell executions

With the new design of Chief, execution of multiple spells simultaneously (especially emergency spells) becomes more difficult due to the `liftCooldown` period.

- In the old Chief contract, multiple spells can be lifted and executed consecutively in a short period.
- In the new Chief contract, a non-zero `liftCooldown` prevents lifting of new spells in several consecutive blocks. To batch multiple emergency spells, one has to put `vote()`, `lift()`, and execution of different spells in a bundle.

6.3 Impact on Multi Emergency Spells Execution

Note **Version 1**

With the new design of Chief, the batched execution of multiple emergency spells becomes more difficult due to the `liftCooldown` period.

- In the old Chief contract, multiple emergency spells can be lifted and executed consecutively in a short period.
- In the new Chief contract, a non-zero `liftCooldown` prevents lifting new spells in several consecutive blocks.

As a result, to batch multiple emergency spells (ES), one has to put `vote`, `lift()`, and `schedule()` of different spells in a bundle to fit into one block, for example:

- bundle action1: vote for ES-A.
- bundle action2: lift ES-A.
- bundle action3: trigger the immediate execution with `ES-A.schedule()`.
- bundle action4: vote for ES-B to exceed ES-A's approvals.
- bundle action5: lift ES-B.
- bundle action6: trigger the immediate execution with `ES-B.schedule()`.

Nevertheless, in practice, this might require non-trivial coordination due to the amount of votes needed and transaction ordering in a block.

Ultimately, batched emergency spell execution as an incident response becomes non-trivial with the new Chief.

6.4 Potential Flashloan Scenarios Still Possible

Note **Version 1**

While the governance timelock gives governance time to drop malicious spells, the flashloan protection mainly serves as a mechanism to prevent the execution of immediately executable actions, called emergency spells. However, emergency spells are limited and can include for example dropping scheduled spells or other emergency actions (e.g. through Mom contracts).

The flashloan protection aims to reduce the risks associated with emergency spells. More specifically, the mechanism implemented disallows all calls to `free()` if there exists some call to `lift()` in the same block before the call to `free()`. This generally implies that an attacker cannot perform emergency spells risk-free.

However, note that this does not prevent all kinds of flashloan attacks. Below is a list of possibilities:

1. **Multiblock MEV**: Some actors might control multiple consecutive blocks. In such cases, the protection mechanism is rendered ineffective due to risk-free trading of the governance tokens over multiple blocks being possible.

2. **Alternative Repayment for flashloan**: Note that the protection mechanism solely prevents sequences `(..., lift, ..., free)` within a block. Hypothetically, if the execution of an emergency spell allows for sufficient value to be extracted in any form, flashloan attacks not involving `free()` could be possible. For example, the following could be possible:

    1. Flashloan funds.
    2. Lock tokens and vote for and lift a malicious emergency spell.
    3. Execute the emergency spell and profit from it (note that if it is not a privileged call, no flashloan would be needed to begin with).
    4. Repay the flashloan with the profits.
    5. Wait one block and free the locked funds.

To summarize, governance should be aware of multi-block MEV and monitor the developments closely. Further, functionality based on `canCall()` should not have extractable value.