PUBLIC

Code Assessment

of the Star Guard Smart Contracts

October 23, 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 Star Guard according to Scope to support you in forming an opinion on their security risks.

Sky implements Star Guard, a module enabling permissionless execution of Star Spells after they were whitelisted by a core spell.

The most critical subjects covered in our audit are access control, functional correctness and the implications for governance operations. The general subjects covered are unit testing, documentation and trustworthiness. Security regarding all the aforementioned subjects is high.

Note that only low severity and informational issues were raised.

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
• Specification Changed	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 Star Guard 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	26 Aug 2025	ef4cd9963deec7958ed48dddb425cd7021445ecd	Initial Version
2	03 Sep 2025	7398ffb283c4490c6e29bea28b92cd57285d4889	After Intermediate Report
3	21 Oct 2025	52239d716a89188b303f137fc43fb9288735ba2e	Release v1.0.1

For the solidity smart contracts, the compiler version `0.8.24` was chosen and `evm_version` is set to `cancun`.

The files in scope were:

```
src/StarGuard.sol
deploy/StarGuardInit.sol
```

2.1.1 Excluded from scope

All other files and dependencies 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 Star Guard, a module enabling permissionless execution of Star Spells after they were whitelisted by a core spell.

2.2.1 Star Guard

The Star Guard module implements an updated mechanism for executing Star Spells. Previously, Star Spells have been directly executed by Core Spells by calling the `SubProxy` directly within the Core Spell. However, the approach reaches its limits quickly due to the maximum block size.

`MCD_PAUSE_PROXY` is expected to remain a ward in the `SubProxy`. Thus, Star Guard offers an additional execution model where Core Spells schedule Star Spells so that they can be executed

permissionlessly in a separate transaction. Note that each Star will have its own Star Guard and that at most one spell can be scheduled per Star Guard. Below is an outline of the new execution model:

1. *Scheduling*: In an initial transaction, core governance (i.e., `MCD_PAUSE_PROXY`) executes their spell which calls `StarGuard.plot` to whitelist a Star Spell on the given Star Guard. If a spell was already scheduled, the new call to `plot` will overwrite it.
2. *Execution*: In another transaction, as long as the maximum delay has not passed since `plot` was called, the spell can be executed by anyone with `StarGuard.exec` if the spell is executable (i.e., `spell.isExecutable` holds) and if the spell's codehash matches the tag provided in `plot` (i.e., codehash provided by the core spell). A successful execution consumes the scheduled spell, meaning it cannot be executed again. Note that the execution calls `SubProxy.exec` with the spell as the target and the selector of `execute()` as the calldata. Note that it is enforced that Star Guard remains a ward in the `SubProxy`.

Also, core governance can unwhitelist whitelisted spells with `StarGuard.drop` as well as configure the maximum delay `maxDelay` with `StarGuard.file`. `StarGuard.prob` serves as a getter to check whether the currently whitelisted spell can be executed. Last, Sky's standard access control with `rely/deny` is implemented.

2.2.2 Deployment

`StarGuardInit.init` implements the initialization of `StarGuard` and is intended to be used within a Core Spell. It performs the following steps:

1. Ensures that `MCD_PAUSE_PROXY` is a ward for `StarGuard` and that the immutable `StarGuard.subProxy` corresponds to the parameter provided.
2. `maxDelay` is set with `file`. However, only a non-zero `maxDelay` is allowed.
3. `StarGuard` becomes a ward in the `SubProxy` so that it can call `SubProxy.exec`.
4. The `StarGuard` and, if necessary, the `SubProxy` are registered in the Chainlog.

2.3 Trust Model

This section outlines the trust model for the contracts in scope. Below, the different roles and trust levels are listed.

- Wards: Fully trusted by the Star. They can force any execution for the `SubProxy`. Expected to be `MCD_PAUSE_PROXY`.
- Sub Proxy: The sub proxy is expected to work correctly. Also, it is expected that only the Star Guard and `MCD_PAUSE_PROXY` are wards in the sub proxy.
- Spells: Fully trusted by the Star. Expected to be validated accordingly (e.g. no malicious actions, no malicious `SubProxy.wards` operations, etc.). Expected to implement the required functions (i.e., `execute` and `isExecutable`).
- Any other address: Untrusted.

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

- Inconsistent Wards Safety Check Specification Changed

Informational Findings	2

- Drop on Init Code Corrected
- Inconsistent Coding Style Code Corrected

6.1 Inconsistent Wards Safety Check

Design Low Version 1 Specification Changed

CS-SKY-SGUARD-003

`StarGuard` ensures that it remains a ward in `SubProxy` after `SubProxy.exec`:

```
subProxy.exec(spellDataCopy.addr, abi.encodeWithSignature("execute()"));
require(subProxy.wards(address(this)) == 1, "StarGuard/subProxy-owner-change");
```

Technically, `MCD_PAUSE_PROXY` should also remain a ward. However, the corresponding check is missing. Similarly, another address could be added as a ward during the spell execution. Verifying this is nearly impossible in the current setup since wards are stored in a mapping. Furthermore, any ward can remove other wards at any time, meaning an unexpected ward could later `deny` both `StarGuard` and `MCD_PAUSE_PROXY`, effectively hijacking control of the `SubProxy`.

Specification changed:

The README now specifies clearly that the intent behind the checks is solely for sanity check purposes. Spells are thus treated as fully trusted and never expected to remove `StarGuard` or `MCD_PAUSE_PROXY` from the `wards` mapping.

6.2 Drop on Init

Informational Version 1 Code Corrected

CS-SKY-SGUARD-002

`StarGuardInit.init` performs several checks regarding the correctness of the deployment. Note that not all properties can be checked in the initialization script, see Deployment verification. However, the initialization script could be more defensive.

More specifically, consider the following example:

1. Star Guard is deployed.
2. The deployer plots a malicious spell.
3. Privileges are handed over to governance as expected.
4. Governance initializes the Star Guard (i.e. giving privileges for `SubProxy`).
5. The malicious spell can now be executed.

Ultimately, governance could be more defensive by calling `drop` so that no such maliciously plotted script can be executed (in case of errors during deployment validation).

Code corrected:

The code has been adjusted to ensure that no address is whitelisted.

6.3 Inconsistent Coding Style

Informational **Version 1** **Code Corrected**

CS-SKY-SGUARD-001

The codebases of Sky typically follow similar style guidelines. The `StarGuard` implementation deviates slightly from the common style for evaluating boolean expression.

More specifically, `prob` and `exec` evaluate boolean expressions as follows:

```
StarSpellLike(spellData.addr).isExecutable() == true
```

Note that, typically, the codebases only specify such statements as `StarSpellLike(spellData.addr).isExecutable()` which is equivalent.

Code corrected:

The `== true` has been removed.

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 Deployment Verification

Note Version 1

Since deployment of the contracts is not performed by the governance directly, special care has to be taken that all contracts have been deployed correctly.

We therefore assume that the initcode, bytecode, traces and storage (e.g. mappings) are checked for unintended entries, calls or similar. This is especially crucial for any value stored in a mapping array or similar (e.g. could break access control which could lead to unexpected contract implementation upgrades and hence result in stealing of funds).

It is of utmost importance that no unexpected wards exist. Upon deployment, `msg.sender` is set as initial ward in the StarGuard contract. It is expected to add `MCD_PAUSE_PROXY` as ward and remove itself. Verification must confirm that after this transition, `MCD_PAUSE_PROXY` is the only ward and no other wards have been added.

7.2 Operational Considerations of Star Guard

Note Version 1

Star Guard introduces an additional execution model for casting Star Spells. Spells executed with Star Guard are not executed as part of the Core Spell transaction but are executed in another one.

Below is a list of considerations for core governance:

- Governance should be aware that ordering of Star Spells is harder to enforce. For example, if the spell for Star X should be executed before the spell for Star Y, the ordering cannot be trivially enforced. Ordering could be possible with designing `isExecutable` accordingly. However, that may not always be possible. As a consequence, plotting might require multiple core spells to enforce ordering.
- Governance should be aware that plotted spells may not be executed. Thus, infrastructure should be run to automatically execute spells when possible (see `prob`).
- Additionally, see Plot Overwriting.
- The on-chain state could change in the window from plotting and execution. As a consequence, the plotted spell may revert or achieve an undesired outcome. Governance should carefully inspect spells to ensure such scenarios do not occur.

Further, note that core governance is expected to keep its privileges in `SubProxy`. In case the `StarGuard` execution model is not suitable for a certain group of Star Spells, the Core Spell may still fall back to directly calling the `SubProxy` (or plotting and immediately executing).

7.3 Ordering of `file` and `plot`

Note Version 1

Governance should be aware the ordering of `file` (i.e., setting the maximum delay) and `plot` may be carefully defined. Namely, a core governance spell first sets the maximum delay and then whitelists a Star Spell, then the deadline will consider the new maximum delay value. In contrast, if the order is vice versa, the deadline would not consider the new maximum delay but rather use the previous value.

7.4 Plot Overwriting

Note Version 1

Governance should be aware that `plot` may overwrite plotted but not yet executed Star Spell.

Consider the following example:

1. Core governance plots a Star Spell in a Star Guard that has a maximum delay of 10 days.
2. The Star Spell returns `false` for `isExecutable` for 7 days (e.g. additional Star Governance Delay implemented in the spell).
3. After 7 days, core governance plots another Star Spell. That effectively replaces the previously whitelisted one.

Thus, core governance should be aware that spells might be replaced if they have not been executed. Similarly, governance should be aware that `plot` could in theory be used as an alternative to `drop` to remove spells from the whitelist.

In Version 3, `plot` now emits a `Drop` event for any existing spell before overwriting it. Additionally, `plot` now requires a non-zero spell address, so it can no longer be used as an alternative to `drop` without replacing with another spell.