PUBLIC

Code Assessment

of the DSS Exec Lib Smart Contracts

October 23, 2026

Produced for

Sky

by

CHAINSECURITY

Contents

1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 9
4 Terminology 10
5 Open Findings 11
6 Resolved Findings 12
7 Informational 15
8 Notes 16

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 DSS Exec Lib according to Scope to support you in forming an opinion on their security risks.

Sky offers DSS Exec Lib, a structured framework for executing governance spells in the Sky Protocol. The library standardizes how spells are defined, deployed and executed while providing a set of functions for managing system parameters, collateral and governance.

Individual spells built using this framework must be reviewed separately; the usage and combinations of DssExecLib functions and the parameter selection when calling these functions are out of scope for this review.

The most critical subjects covered in our audit are functional correctness, NatSpec documentation and operational usability. Several observations are highlighted as notes or informational issues in this report.

The general subjects covered are code quality, documentation, maintainability, and correctness of existing functionality. Note that we reviewed library completeness at a high level only, focusing primarily on verifying that existing functions work correctly. For detailed considerations regarding completeness, see Completeness of Functionality.

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 DSS Exec Lib 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	01 Sep 2025	19ea58b332058b424ddf4db6a0eba5944d12ff96	Initial Version
2	12 Sep 2025	3996d7083f66bf449fd55ccca88cf20963dac1b4	After Intermediate Report
3	05 Jun 2026	555d1c158f2ab695e04d6de6e603b118d32214c4	StarGuard Addition
4	08 Oct 2026	de6b715c12016d008735301ce0d7c2116162f1ac	Kicker and AutoLine
5	16 Oct 2026	1bc3c3747e562bbbfdfbbc2fdd178aa85665db20	AutoLine Precision Check
6	23 Oct 2026	4206c58c162116e211067bbe3a3d3aae7eda994d	AutoLine Ordering Documentation

For the solidity smart contracts, the compiler version `0.8.16` was chosen and `evm_version` is set to `cancun`.

The files in scope were:

```
CollateralOpts.sol
DssAction.sol
DssExec.sol
DssExecLib.sol
```

2.1.1 Excluded from scope

All other files and dependencies are out of scope. Spells built using DssAction and the DssExecLib library are excluded and must be reviewed separately. The usage, combinations and parameter selection of DssExecLib functions are also out of scope and assumed to be done correctly. Additionally, note that we expect that the functions provided in DssExecLib are the expected functions (note that some might be missing or some could be outdated).

2.2 System Overview

This system overview describes the latest version (Version 5) 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 offers DSS Exec Lib, a structured framework for executing governance spells in the Sky Protocol. The library standardizes how spells are defined, deployed and executed while providing a rich set of functions for managing system parameters, collateral and governance. DssExecLib was extended for Sky Protocol with new token support (USDS, sUSDS, SKY), rate management functions (SSR), SubDAO (star) spell execution, helpers for StarGuard, and setters for the Kicker surplus parameters (Version 4).

2.2.1 DssExec

The spell deployment wrapper. A spell is deployed as follows:

```
new DssExec(
    "A test dss exec spell",       // Description
    block.timestamp + 30 days,     // Expiration
    address(new SpellAction())
);
```

Encapsulates description, expiration, and reference to the action contract.

Exposes `schedule()` to queue the spell for later execution in `MCD_PAUSE`. The spell must not be expired at the time of scheduling and the contract must be authorized via `MCD_ADM`: SKY token holders vote in `Chief` and once this spell is elected as the `hat`, it has the privilege to call `plot()` in `MCD_PAUSE`.

Provides `cast()` to trigger execution via `MCD_PAUSE`. This can be called permissionlessly once conditions are met and marks the spell as executed. Execution may also be triggered directly in `MCD_PAUSE`, bypassing `cast()`.

2.2.2 DssAction

Abstract contract defining the structure of a spell.

A spell must inherit `DssAction`. Developers must override the `actions()` function and may use functionality of the library `DssExecLib` for protocol interactions.

This provides a uniform interface for governance execution. Most importantly, that includes `execute()` as the entrypoint (which utilizes `actions()` and limits according to the office hour configuration).

2.2.3 DssExecLib

The core library that exposes governance actions as simple function calls. It abstracts low-level interactions with protocol contracts and enforces precision conventions (amounts, rates, durations, percentages). It includes:

- **Core address helpers**: resolve common system contracts (e.g. `vat()`, `jug()`, `pot()`, `dog()`, `kicker()`).
- **Risk parameter setters**: adjust global and per-ilk debt ceilings, stability fees, liquidation ratios, auction parameters and surplus parameters.
- **Rate accumulation**: update DSR, SSR, and collateral stability fee accruals.
- **Collateral onboarding and management**: helper routines for adding new collateral types with pre-defined `CollateralOpts` and helpers for managing configurations.
- **Changelog (Chainlog) management**: update on-chain changelog addresses, versions, and metadata.

- **Governance and SubDAO helpers**: manage authorizations, office hours, GSM delay, whitelist and `file()` patterns, execute SubDAO (star) spells, and manage their whitelisting in the corresponding `StarGuard` via `plotStarSpell` / `dropStarSpell`.
- **Misc**: Various other functions related to configuring interpolations, updating oracles and more.

2.2.4 CollateralOpts.sol

Defines a standard struct for collateral configuration when onboarding new assets.

- Identifiers (`ilk`, `gem`, `join`, `clip`, `calc`, `pip`)
- Risk parameters (debt ceiling, min vault size, liquidation ratio, penalty, max liquidation size)
- Auction parameters (starting price factor, duration, permitted drop, keeper incentives)
- Stability fee and liquidation toggles

2.2.5 Changelog

In Version 3, the StarGuard helpers (`plotStarSpell / dropStarSpell`) were added, along with stricter sanity checks for `setIlkMaxLiquidationAmount` and `setKeeperIncentiveFlatRate`.

In Version 4, the following changes were made:

- Removed `govGuard()`, `flip()`, `setDDMTargetInterestRate` and the Vow surplus setters `setSurplusAuctionAmount` / `setSurplusBuffer`.
- Added the Kicker helpers `kicker()`, `setKickerAuctionAmount` (`kbump`) and `setKickerSurplusBuffer` (`khump`).
- `setIlkAutoLineParameters` and `setIlkAutoLineDebtCeiling` now call `exec()` on `DssAutoLine`, so that the ilk `line` and the global `Line` are updated immediately.
- The debt auction helpers were intentionally left unchanged, as it is not yet defined whether the currently disabled debt auction mechanism will be removed, adapted to the `Kicker` architecture or replaced.

In Version 5, `setIlkAutoLineDebtCeiling` enforces the same `_amount < WAD` precision check as `setIlkAutoLineParameters`.

2.3 Trust Model

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

1. Governance
    - The Governance (Sky Token holders / voters) schedule the spell in `MCD_PAUSE` and the spell executes via the `MCD_PAUSE_PROXY` after the mandated governance delay and any office hours restriction.
    - Trust Level: Fully trusted
    - Worst Case: Can change any governed parameter.
2. Spell Author and Deployer
    - Description: Writes the DssAction implementation and deploys DssExec with the action address, description, and expiration.
    - Trust Level: Untrusted. Code must be inspected prior to scheduling the spell.
3. Any External Caller

- Description: Can call `schedule()` and `cast()` when conditions allow. Can call `actions()` directly on the action contract, but without pause proxy authority.
- Trust Level: 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:

- 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	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

- FlipperMom Missing in Chainlog Code Corrected

Informational Findings	3

- Autoline ttl Check Inconsistency Code Corrected
- NatSpec Problems Specification Changed
- README Inaccuracies Specification Changed

6.1 FlipperMom Missing in Chainlog

Correctness Low Version 1 Code Corrected

CS-SKY-EXL-001

`DssExecLib` implements

```
function flipperMom() public view returns (address) {
    return getChangelogAddress("FLIPPER_MOM");
}
```

At the time of this review the key `FLIPPER_MOM` is not present in the chainlog, using this function will revert.

Code corrected:

This getter was removed from DssExecLib.

6.2 Autoline `ttl` Check Inconsistency

Informational Version 1 Code Corrected

CS-SKY-EXL-002

`DssExecLib.setIlkAutoLineParameters()` (the variant without the `ttl` parameters) retrieves and reuses the existing `ttl` value without validating it's non-zero.

In contrast, `setIlkAutoLineDebtCeiling()` includes a `require(gap != 0 && ttl != 0)` check.

This inconsistency means `setIlkAutoLineParameters()` can operate on not configured ilks (where `ttl == 0`), potentially leading to unintended configurations, while `setIlkAutoLineDebtCeiling()` would revert in the same scenario.

Code corrected:

`setIlkAutoLineParameters()` was updated to include a sanity check ensuring the retrieved `ttl` is non-zero.

6.3 NatSpec Problems

Informational **Version 1** **Specification Changed**

CS-SKY-EXL-003

While NatSpec is provided, some problems exist. Below is a non-exhaustive list of problems related to NatSpec:

1. Several functions do not have any NatSpec associated with them (e.g. getters such as `vat()`).
2. `setAuthority`: The NatSpec is inaccurate. It should specify that the authority is set and that `_authority` corresponds to the address managing access control. However, it specifies that authority will have privileges to perform actions. While that could be possible, the authority will be able to manage privileged access (e.g. Chief will manage who can call the Pause contract).
3. `setRWAIlkDebtCeiling`: Specifies `bytes32("ETH-A")` as an example which might be a misleading example.
4. `addNewCollateral`: Lacks NatSpec.
5. `addCollateralBase` & `addNewCollateral`: Lack descriptions of the intended usecases as the functions are not always suited to be used independently.
6. `executeStarSpell`: "Execute a start spell" has a typo. It should be "Execute a star spell".

Specification changed:

The NatSpec comments in the code have been updated to address the issues listed above.

6.4 README Inaccuracies

Informational **Version 1** **Specification Changed**

CS-SKY-EXL-005

The README documents the functions. However, there are some inaccuracies. Below is a non-exhaustive list of the inaccuracies:

1. Core Address Helpers: `esm()` is undocumented but in code.
2. Core Address Helpers: `govGuard()` is documented but in code the function is called `mkrGuard()`.
3. `getChangelogAddress`: The function is in section "Changelog Management" in the README. In code, it is in section "Core Address Helpers".
4. System Configuration: The section lacks the documentation of the two `setValue()` functions.
5. `setSurplusAuctionMinPriceThreshold`: `9_80` is suggested for a 2% drop. However, it should be `98_00`.

6. Abacus Management: The functions are documented as `init*Decrease()`. However, in code they are named `set*Decrease()`.
7. Collateral Onboarding: `kprFlatReward`, `kprPctReward` and `breakerTolerance` missing in CollateralOpts example and are not mentioned in the section.
8. SubDAO/Star Spells: The code for these is undocumented in the README. Thus, `executeStarSpell` and `tryExecuteStarSpell` are not documented.

Specification changed:

The README has been updated to resolve most issues listed above:

- The function `mkrGuard` has been renamed to `govGuard` in code, with the README updated to read SKY instead of MKR Authority. Note `govGuard` has been removed in Version 4.
- The `getChangelogAddress` location discrepancy remains unchanged.

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 Non-zero `jug.base`

Informational **Version 1** **Risk Accepted**

CS-SKY-EXL-004

`DssExecLib.setGlobalStabilityFee` allows setting `jug.base`. In previous audits and discussions, this value was assumed to always remain 0; a non-zero value would break certain systems (e.g. Sky stUSDS).

This function's presence in the library may suggest it is a valid option to use and hence increases the risk of mistakes when crafting spells. There is no warning that such usage is unexpected.

Risk accepted:

Sky states:

```
This is one of the functions that should be marked as deprecated for removal in
future versions. For now it is sufficient to say that governance is aware this
method should not be used.
```

7.2 Require Error Messages

Informational **Version 1** **Acknowledged**

CS-SKY-EXL-006

No require statement has an error message. However, they are written as in the example below:

```
require(_eta != 0); // "DssExecLib/invalid eta"
```

The reason could be the age of the codebase. However, since the library was modernized to Solidity 0.8, `require` statements with error messages could be used.

Acknowledged:

Sky states:

```
This is by design. The require error messages are not included to save on contract
size, since the library is already pretty close to the limit without error messages.
```

8 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.

8.1 Completeness of Functionality

Note Version 1

Users of the library as well as governance should be aware that `DssExecLib` only partially covers the functionality that governance can call. Thus, the library should only be seen as a helper but is not expected to be complete in terms of functions available.

Below, we list examples of such functionality:

1. Getters: Many getters for various contracts (e.g. `vat()`) exist. Some seem to be missing. `MCD_SPLIT` is a good example since the `flap` function exists. Another example of a contract without a getter is `SPBEAM_MOM`. To summarize, not all possible getters are provided.
2. For some functionality it is unclear whether there could be more versions of it (e.g. `file`). Note that wrappers around such functions might not be fully provided.
3. For some system parameters there is custom logic wrapping `file`. For example, the `vow` can be set in many contracts but no wrapper for setting the `vow` is available for each such contract.
4. The system is big and it is not expected that all functionality is covered. For example, the DDM related functions only include `setDDMTargetInterestRate` which is very specific to one plan. Other plans, pools, and the hub have more configuration possibilities. As of Version 4, `setDDMTargetInterestRate` has been removed and the example no longer applies, but similar cases exist for other parts of the system.
5. Various system operations may come in different flavors. Most notably that includes collateral onboarding where `addNewCollateral` is not suitable for all collaterals. For example, Lockstake requires custom scripts. Users should be aware that every usage of every function must be carefully thought through to ensure the correctness of Spells.

To summarize, the library is not expected to be complete. That is due to the size of Sky's ecosystem, number of parameters and variations in contracts. Users should be aware and thus use the library responsibly and adjust their code accordingly.

8.2 Hole and Dust Setter Considerations

Note Version 1

The `hole` of an ilk in the Dog and the `dust` of an ilk in the Vat are linked by the invariant `hole >= dust`. Both setters enforce it, each checking the new value against the *current* stored value of the other:

- `setIlkMinVaultAmount` (sets `dust`) requires `new dust <= current hole`.
- `setIlkMaxLiquidationAmount` (sets `hole`) requires `new hole >= current dust`.

Consequently, when both are updated for the same ilk within a single spell, the required call order depends on the direction of the change:

1. When raising both, `setIlkMaxLiquidationAmount` (hole) must be called first.
2. When lowering both, `setIlkMinVaultAmount` (dust) must be called first.

The above has been updated to reflect Version 3. Prior to that, `setIlkMaxLiquidationAmount` did not enforce `new hole >= current dust`, so only `setIlkMinVaultAmount` guarded the invariant and the ordering consideration was one-sided (lack of sanity check was reported as part of this note).

8.3 Manipulations May Lead to Skipping Intended Execution

Note Version 1

The following functions perform low-level calls:

1. `tryExecuteStarSpell`: To execute `starProxy.exec(spell, "execute()")`. That is to allow governance spells to handle errors in greater details instead of simply reverting as `executeStarSpell` would do.
2. `setIlkMinVaultAmount`: To execute `clipper.upchost()`. That is due to the clipper being `0x0` for some ilks (catching such errors).
3. `setIlkLiquidationPenalty`: As in 2.

Note that the low-level calls could be manipulated to fail. Examples of potential manipulations could include:

- Gas manipulations leading to Out-of-Gas errors. For example, the executor could provide exactly enough gas so that the final call executes but fails with Out-of-Gas, while the remaining gas (recall that the EVM only forwards up to 63/64 of the available gas) is still sufficient to complete the transaction.
- Other manipulations (e.g. triggering reentrancy locks).

Governance should be aware that such possibilities could exist and should ensure that such manipulations are not possible or handled accordingly in the spells using the given functionality. Otherwise, outcomes may not be as expected.

8.4 Office Hours

Note Version 1

Office hours management in the *dss_exec* library operates on Unix time. By convention, office hours are Monday through Friday from 14:00 to 21:00 UTC.

Caveats:

- **Leap seconds** Unix timestamp counts elapsed seconds since 1970-01-01 00:00:00 UTC, treating every day as exactly 86400 seconds. Real UTC occasionally includes leap seconds (rare days with 86401 seconds). As a result, Unix time drifts from astronomical UTC by a few dozen seconds. In practice this means the 14:00–21:00 UTC window is defined according to Unix time, not true astronomical UTC, and may be offset by a few seconds in real UTC.
- **Daylight savings** UTC does not observe daylight savings time. Office hours are fixed in UTC. When viewed in a local timezone that observes DST, the apparent local office hours will shift by one hour twice per year.

8.5 Setting `mat` Does Not Lead to `poke`

Note Version 1

Users of the library should be aware that `setIlkLiquidationRatio`, the setter for an ilk's liquidation ratio `mat` in the spotter, does not automatically call `poke` on the spotter to propagate the changes. However, that is typically expected to occur.

However, note that `addNewCollateral` propagates the price to the Vat by manually invoking the function (as both `pip` and `mat` are set).

8.6 Star Spell Plot and Drop Edge Cases

Note **Version 3**

`plotStarSpell` and `dropStarSpell` are thin wrappers around the `StarGuard`'s `plot()` and `drop()` functions. They forward their arguments without additional checks:

1. `plotStarSpell` does not check whether a star spell is already plotted in the `StarGuard`. If one already is, calling `plot()` again overwrites it, effectively dropping the previously plotted spell.
2. `dropStarSpell` calls `drop()` unconditionally. If no spell is currently plotted, the `StarGuard` does not revert and the call is a silent no-op. (In `plotStarSpell` a zero spell address would instead be detected within `StarGuard`.)

This is intentional. Re-plotting is a legitimate operation (e.g. replacing a plotted spell before execution) and `drop()` is idempotent. The plotted spell is readable on-chain, so spell simulation can surface these conditions directly against `StarGuard`.

8.7 done Flag May Be Unreliable

Note **Version 1**

DssExec provides `cast()` to trigger the execution via `MCD_PAUSE`:

```
function cast() public {
    require(!done, "spell-already-cast");
    done = true;
    pause.exec(action, tag, sig, eta);
}
```

Execution may also be triggered directly in `MCD_PAUSE`, bypassing `cast()`. In this case the `done` flag will not be set to `true` when a spell has been executed.

Sky states:

```
The infrastructure that TechOps built for scheduling and casting actually depends
on this. There's this notion of "conforming spells", which are basically spells
that inherit from the DssExec base class.

The assumption is that any governance-supported spell should be conforming.

However, you are correct to point out that it is not actually required to go through
that method. Once the spell has the hat, anyone could call MCD_PAUSE.exec directly.
However, there are neither incentives nor further consequences in doing so.
```

8.8 `setIlkDebtCeiling` and Autoline

Note Version 1

`DssExecLib.setIlkDebtCeiling`, `increaseIlkDebtCeiling` and `decreaseIlkDebtCeiling` do not take autoline configurations into account. Using this without careful consideration can lead to unwanted outcomes as autoline may be active and interfere.