PUBLIC

Code Assessment

of the SP-BEAM Module Smart Contracts

March 31, 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 SP-BEAM Module according to Scope to support you in forming an opinion on their security risks.

Sky implements SP-BEAM, a module enabling permissioned actors to make direct changes to stability and savings rates.

The most critical subjects covered in our audit are functional correctness, access control, and integration with the existing contracts. The general subjects covered are documentation, trustworthiness, and unit testing.

Security regarding all aforementioned subjects is high. However, it is improvable due to potential escalation of privileges as outlined in Bypassing step-size and DoSing ilk initializations.

In summary, we find that the codebase provides a good but improvable 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	3
• Code Corrected	2
• Risk Accepted	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 SP-BEAM Module 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	07 Mar 2025	1d08161538f7ae9869de535803b5229e99fd6112	Initial Version
2	19 Mar 2025	c9dfdfb5c40f15885c425ab64188f3544412c1c8	After Intermediate Report
3	28 Mar 2025	4e7a435682bf0a4fb9f19070f5ff34ffbdee6d9d	Renaming

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

The following files were in scope:

```
src/
    SPBEAM.sol
    SPBEAMMom.sol
    deployment/
        SPBEAMInit.sol
        SPBEAMInstance.sol
        SPBEAMDeploy.sol
```

2.1.1 Excluded from scope

Generally, all files not mentioned above are out of scope. Further, note that it is assumed that both, Pot and SUsds, are initialized and have a rate (`dsr` or `ssr`) greater than or equal to `RAY`. Further, it is assumed that `Jug.base == 0` holds.

2.2 System Overview

This system overview describes the latest version of the contracts as defined in the Assessment Overview. At the end of this report section, we have added a subsection outlining the most significant changes introduced in each version.

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 Stability Parameter Bounded External Access Module (SP-BEAM) which enables permissioned actors to make direct changes to stability parameters (`ilk.duty`, `POT.dsr`, `SUSDS.ssr`) respecting the defined limits.

2.2.1 SPBEAM

The SPBEAM contract provides the typical administrative functions to add/remove admins (`rely()` / `deny()`) and to whitelist/remove privileged actors (`kiss()` / `diss()`).

The admins can configure the parameters with:

- `file(bytes32 what, uint256 data)`: sets the cooldown period `tau`, pausing status `bad`, or the last update timestamp `toc`.
- `file(bytes32 id, bytes32 what, uint256 data)`: sets the configuration (`min`, `max`, `step`) for a given rate identifier `id` (the ilk name, "DSR", or "SSR").

The whitelisted actors can trigger parameter changes in a batch with `set()` when the SPBEAM is not paused (`bad == 0`) and the cooldown period `tau` has elapsed. For each update in the batch:

- The new yearly rate in basis points (BPS) should be within the `min` and `max` limit.
- A converter's function `conv.rtob()` is used to convert the old rate per second to a yearly rate in BPS.
- The difference between the new and the old yearly rate in BPS should be within the `step`. Note that if the old yearly rate is lower or higher than the `min` and `max`, the BPS value used is the `min` and `max`, respectively.
- The new rate per second in RAY is computed with the converter's function `conv.btor()`.
- Eventually it updates the accrued interest with `drip()` on respective contracts before changing the rate with `file()`:
    - For POT it changes `dsr`: `pot.file("dsr", ray)`.
    - For SUSDS it changes `ssr`: `susds.file("ssr", ray)`.
    - For ilks it changes `duty`: `jug.file(ilk, "duty", ray)`.

The configuration for each `id` can be retrieved with the getter `cfgs()`.

2.2.2 SPBEAMMom

SPBEAMMom is a helper contract to halt the SPBEAM module without enforcing the GSM delay in emergency. `owner` is the admin of the contract, it can either change the ownership (`setOwner()`), or change the authority address (`setAuthority()`).

The emergency action `halt(address spbeam)` can only be triggered by authed parties. It will file the `bad` status in the SPBEAM contract to `1` in order to pause it. Note that this is typically expected to occur when a corresponding emergency spell is elected as the `hat` within DSChief.

2.2.3 Deployment Scripts

The library SPBEAMDeploy provides the function `deploy()` that creates a new SPBEAM contract and a SPBEAMMom contract. The wards of SPBEAM and owner of SPBEAMMom are immediately switched to an input owner (assumed to be the DSPauseProxy). A SPBEAMInstance containing the SPBEAM and SPBEAMMom is returned.

The library SPBEAMInit provides the function `init()` that initializes the SPBEAMInstance with the given parameters. It is assumed to be executed in the context of the DSPauseProxy:

- It grants `wards` role of SPBEAM to SPBEAMMom.
- The `authority` of SPBEAMMom is set to DSChief ("MCD_ADM").

- It grants `wards` role of JUG, POT, and SUSDS to SPBEAM.
- It sets the cooldown period `tau`.
- An array of ilk configurations is set in SPBEAM.
- A facilitator is enabled with `kiss()`.

2.2.4 Changelog

In Version 2, the following changes were introduced:

- `toc` is settable with `file()`. By setting `toc` to a timestamp in the future, it prevents the module from being used for a limited period of time if required
- The previous annual rate is bounded by `min` and `max`.

In Version 3, the following changes were introduced:

- DSPC (Direct Stability Parameters Change Module) was renamed to SP-BEAM.

All occurrences of "DSPC" were replaced with "SPBEAM" in the report to reflect the latest naming convention.

2.3 Trust Model

Contract SPBEAM features the following roles:

- **wards**: admin of the contract; assumed to be the DSPauseProxy and the SPBEAMMom.
    - **Trust Level**: Fully trusted.
    - **Worst-case**: The wards can update the configurations to change the stability parameters to arbitrary value (i.e. inflating the savings rate / stabilitiy fees).
- **buds**: actors who can trigger the parameter changes.
    - **Trust Level**: Partially trusted.
    - **Worst-case**: The buds can change the stability parameters to value within the configured limits and potentially arbitrage. In the event of compromised buds, the wards can remove them from the whitelist.

Contract SPBEAMMom features the following roles:

- **owner**: admin of the contract; assumed to be the DSPauseProxy.
    - **Trust Level**: Fully trusted.
    - **Worst-case**: The owner can halt the SPBEAM module, or change the authority to arbitrary contracts.
- **authority**: contract that specifies authed parties who can trigger the emergency halt; assumed to be DSChief.
    - **Trust Level**: Fully trusted.
    - **Worst-case**: Any allowed parties on authority can halt the SPBEAM module.

In addition,

- **Deployer**: untrusted; deploys the contract using the SPBEAMDeploy script. Could modify the script and/or state of the deployment contract before transferring ownership.

- **Governance**: fully trusted; will trigger the execution of the SPBEAMInit code executed as delegatecall in the context of the DSPauseProxy. Must ensure the parameters passed and the code to be executed is correct.

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:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- 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	1

- BPS Changes Without Effect Increases Toc Risk Accepted

5.1 BPS Changes Without Effect Increases Toc

Correctness Low Version 1 Risk Accepted

CS-MK-SPBEAM-001

To prevent `toc` updates without presenting any updates, the following check on `updates` is performed:

```
require(updates.length > 0, "SPBEAM/empty-batch");
```

However, the check can be circumvented by providing `bps` values equal to the prior values so that `set.delta == 0`. As a consequence, no changes would be applied while `toc` would be increased.

While such a batch in theory is not empty, no changes are applied. Consequently, the empty-batch check is circumvented.

Risk accepted:

Sky is aware and accepts the risk.

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	2

- Bypassing Step-Size Code Corrected
- DoSing Ilk Initializations Code Corrected

Informational Findings	1

- Unsuitable Parameter Settings Code Corrected

6.1 Bypassing Step-Size

Correctness Low Version 1 Code Corrected

CS-MK-SPBEAM-002

The configuration parameter `step` aims to limit the capabilities of a facilitator. More specifically, it enforces a limit on the delta of the annual rate in BPS for a given call to `set()`.

However, malicious facilitators could circumvent the checks by passing a parameter `updates` with repetitive IDs.

Consider the following example:

1. `_cfg["dsr"].step = 10_00` while the current annual rate corresponds to `10_00`. `_cfg["dsr"].max = 30_00` corresponds limits the maximum annual rate to 30%.
2. Thus, it is expected that two calls to `set()` are required to reach the maximum.
3. `set()` is called with `updates = [("dsr", 20_00), ("dsr", 30_00)]`.
4. The first iteration validates that the delta is valid according to the step-size (`20_000-10_000 <= 10_000`).
5. The second iteration validates that the delta is valid according to the step-size (`30_000-20_000 <= 10_000`).
6. As a consequence, the maximum is reached within one call to `set()`.

To summarize, the checks against `step` can be bypassed by providing repetitive IDs.

Code corrected:

IDs are now required to be ordered in ascending order. Hence, duplicated elements are prevented. Note an update exceeding `step` is still possible in Version 2, since it adjusts the old rate to `min` or `max` if it is out of bounds in an update.

6.2 DoSing Ilk Initializations

Security | Low | Version 1 | Code Corrected

CS-MK-SPBEAM-003

Governance spells can be DoSed by facilitators by initializing ilks on the Jug in unintended ways.

For context, spells enabling certain ilks follow a common procedure where an `ilk` is initialized in the Vat and then in the Jug (see example):

```
dss.vat.init(ilk);
dss.jug.init(ilk);
```

Note that the initialization within the Jug performs the following:

```
function init(bytes32 ilk) external note auth {
    Ilk storage i = ilks[ilk];
    require(i.duty == 0, "Jug/ilk-already-init");
    i.duty = ONE;
    i.rho  = now;
}
```

Thus, the call will only be successful if and only if the `duty` is zero.

The SPBEAM allows working on uninitialized ilks. More specifically,

- if the config has not been set, the only valid `bps` value is 0.
- if the config has been set, `bps` could also be above zero. However, depending on the configuration, calls to `set()` might also revert.

The call to `jug.drip()` will

- set `rho` to the current block timestamp within the Jug.
- set `rate` to a non-zero value.
- not change any state in the Vat if the ilk has not been initialized there or change state accordingly if the ilk has been initialized within the vat.

Now, assume a scenario where an ilk is scheduled to be initialized within a governance spell. A facilitator can effectively break the spell by performing `set()` for the to-be-initialized ilk.

To summarize, ilk initializations can be frontrun by malicious facilitators to DoS scheduled governance spells.

Code corrected:

The code has been adjusted to enforce that a configuration for an ilk has been provided by validating that `cfg.step > 0` holds. Note that `file()` now requires that `duty > 0` holds when configuring ilks.

6.3 Unsuitable Parameter Settings

Informational | Version 1 | Code Corrected

CS-MK-SPBEAM-004

Note that some parameter configurations might not allow for simple `set()` operations.

Consider the following example:

1. The SSR is set to 10%.
2. The min and the max are configured to 12% and 14% while the step size is 1%.
3. `set()` will revert for any BPS value provided for the SSR.

However, note that it could make sense to allow setting the SSR to 12% directly in such cases. Similarly, this could be applied in the other direction.

Ultimately, some settings could potentially require some bigger jumps due to the minimum or maximum being unreachable through the regular step-size.

Code corrected:

The code now bounds the old annual rate to `min` and `max`.

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 Asymmetrical Risks Between Rate Increase and Decrease

Note Version 1

The updates of all rates share a same global frequency limit `toc`, namely a subsequent batch updates should wait `tau` after a prior batch updates, regardless of the `ids` in the batch.

However, different `ids` have different risk profiles. For instance, in general, increasing the rate of SSR or DSR will increase the system risks (less system surplus), whereas increasing the rate of stability fees will decrease the system risks (more system surplus).

Consequently, the system risks may in general increase or decrease in a batch update. The facilitator should be careful when preparing a batch updates, since decreasing the system risk is also subject to the frequency limit.

7.2 Considerations for Configurations

Note Version 1

Below is a list of considerations regarding the configurations of ilks:

- If a new minimum would be greater than the currently set maximum, the new maximum must be set first.
- If a new maximum would be less than the currently set minimum, the new minimum must be set first.
- For a config initialization, setting the maximum, the minimum and the step should be performed atomically within one transaction (as outlined in the initialization script). Note that this is mainly related to the step size. If only the step-size were to be initialized in a prior transaction, an unwanted reduction to `bps==0` could be possible.

7.3 No Ilk With Name "DSR" / "SSR"

Note Version 1

Governance should be aware that the SP-BEAM implementation requires that an ilk will never be called `SSR` or `DSR`. Otherwise, the `duty` of such an ilk cannot be adjusted by SP-BEAM.

7.4 Rate Converter Considerations

Note Version 1

The rate converter `conv` will provide crucial functionality. Below are some considerations regarding the converter:

- `btor()`: Assumed to compute the per-second rate in RAY for a given annual rate in basis points. Assumed to never return a value <= RAY. Note that since Version 2 this is explicitly enforced by requiring that `btor(bps) >= RAY`.
- `rtob()`: Assumed to compute the annual rate in basis points for a given per-second rate in RAY.
- Both should be implemented correctly.
- Last, `rtob()` and `btor()` implement checks for the passed in parameters. Governance, should be aware that values are restricted by the constraints of the converter. Additionally, note that no sanity checks are performed within the SP-BEAM.