PUBLIC

Code Assessment

of the Sky Smart Contracts

June 06, 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 Sky according to Scope to support you in forming an opinion on their security risks.

Sky implements SKY, a rebranded version of MKR, and converters with a fixed conversion rate between MKR and SKY. This audit report reviews the security and correctness of both the contracts and the corresponding deployment scripts.

The most critical subjects covered in our audit are security, functional correctness and seamless integration with the existing system. Security regarding all the aforementioned subjects is high.

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 Sky repository.

The scope consists of the two solidity smart contracts:

```
src/
    MkrNgt.sol
    Ngt.sol
```

In Version 2 the deployment scripts were added to scope:

```
deploy/
    NgtDeploy.sol
    NgtInit.sol
    NgtInstance.sol
```

As of Version 5, the files have been renamed as a result of a rebranding. The files below are in scope:

```
src/
    MkrSky.sol
    Sky.sol
deploy/
    SkyDeploy.sol
    SkyInit.sol
    SkyInstance.sol
```

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	11 May 2023	f05f38322aa9910e2e708157c27aa251db0bd72d	Initial Version
2	18 Sep 2023	3830c77d1d9ecbd060a3a2925d46574376f694d8	Deployment Scripts
3	10 Oct 2023	664e1f13f20e9469bda5765b2ac5d1fa6b662b77	Deployment Scripts Fixes
4	16 Oct 2023	2d675f802ac00ed2fb436d5311da415fdb774b50	Allowance Simplification
5	27 Aug 2024	3ddfa5ee55d10b8192f239235bf83e46744314b8	Renaming
6	11 Sep 2024	3e1d92f90c5e2a1ff803809d59fd0adaec9d3b80	L2 Token
7	10 Apr 2025	893bccd61fe28b64dedd00b16574d2b54122da3c	One Direction Converter
8	03 June 2025	e91dfb7923e93e8b543ae464461fb0f2a17ff00e	Disable Legacy Converter

For the solidity smart contracts, the compiler version `0.8.16` was chosen. Since Version 5, the compiler version `0.8.21` is used. In the latest reviewed version (Version 7) the `evm_version` is set to `shanghai`.

2.1.1 Excluded from scope

Any other file not explicitly mentioned in the scope section. In particular tests, external dependencies, and configuration files are not part of the audit scope.

2.2 System Overview

This system overview describes the latest received version as well as the previously deployed version (Version 6) of the contracts as defined in the Assessment Overview.

At the end of this report section we have added a changelog for each of the changes accordingly to the versions.

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 a new governance token SKY, a rebranded version of MKR, with converters that facilitate the conversion between MKR and SKY at a fixed rate (1 MKR:24000 SKY):

- The bidirectional converter (Legacy MkrSky, up to Version 6) allows users to convert MKR to SKY and vice versa.
- The unidirectional converter (MkrSky, added in Version 7) only allows users to convert MKR to SKY with a fee that can be set by the wards.

2.2.1 SKY

SKY is an ERC-20 compliant token with 18 decimals. The contract is controlled by privileged roles `wards` initialized with `msg.sender` in the constructor. Any address in `wards` has privileged access to:

- Add a new ward with `rely()`.
- Remove a ward with `deny()`.
- Mint any amount of SKY tokens to an address with `mint()`.

Token transfers work the same way as a normal ERC-20 token but with a few restrictions. Specifically, transfers to the zero address (`address(0)`) or the contract itself are not allowed. A user can also burn its tokens by calling `burn()` with its own address. In case the address specified is different from the `msg.sender`, the user will burn on behalf of others if its allowance is sufficient.

SKY supports unlimited allowance by approving `max(uint256)`. In addition, `permit()` is provided for setting allowance with signatures either from an EOA or a contract (EIP-1271). A contract can give permission to a spender by implementing `isValidSignature()` with customized verification logic. If the signature length does not equal to 65 bytes, it is assumed the allowance owner is a contract, which will be queried for signature validation.

2.2.2 Bidirectional Converter

Note that this subsection only describes the repository up to Version 6.

Legacy MkrSky is an immutable and permissionless bidirectional converter for SKY and MKR tokens with a fixed conversion rate, which is expected to be 1 MKR to 24000 SKY (`rate=24000`). It only provides two functions:

- `mkrToSky()`: Burns `msg.sender`'s MKR tokens (`mkrAmt`) and mints SKY tokens(`mkrAmt*rate`) to the specified receiver.
- `skyToMkr()`: Burns `msg.sender`'s SKY tokens (`skyAmt`) and mints MKR tokens (`skyAmt/rate`) to the specified receiver. This rounds down if `skyAmt` is not a multiple of `rate`.

To achieve the functionalities above, MkrSky requires the following privileges:

- Be authorized by the MKR contract. This is required for MKR minting.
- Be a member of wards of SKY contract for SKY minting.

Users must give allowance to the contract for the tokens to be burned successfully.

In addition, although MkrSky is non-stoppable, functions above could still revert if the MKR contract is paused (stopped in the contract's terminology).

2.2.2.1 Deployment of Sky and Bidirectional MkrSky

In general, the contracts are deployed in two steps:

1. Some EOA deploys the contracts and - if necessary - changes the owner of these contracts to the `PauseProxy`.
2. A governance `Spell` with quorum executes the initialization of the contracts through the `PauseProxy`.

After deployment, the owner of the `Sky` contract is changed to the `PauseProxy`. `MkrSky` is then deployed with the address of the `Mkr` token, the newly deployed `Sky` address and a rate that indicates the exchange rate between Sky and MKR.

During the initialization of the contracts, the deployment parameters are checked for validity. `MkrSky` is added as owner to both `Sky` and `Mkr` contracts.

Finally, the addresses are added to the chainlog.

In the deployed MkrSky, the MKR:SKY rate is set to 24000.

2.2.3 Unidirectional Converter

The new MkrSky (implemented in Version 7) is a unidirectional converter, different from the Legacy MkrSky, it only supports the one-way conversion from MKR to SKY with an immutable rate.

The standard `wards` authorization mechanism with `rely()` and `deny()` is used. And the following functions are implemented:

- `mkrToSky()`: Burns `msg.sender`'s MKR tokens (`mkrAmt`) and transfers SKY tokens (`mkrAmt*rate-fee`) to the specified receiver. The fee (`<=100%`) is 0 upon deployment and can be configured by the `wards` with `file()`.
- `collect()`: `wards` can collect (`transfer`) the fees accumulated to a recipient.
- `burn()`: `wards` can burn a specified amount of SKY tokens held by this contract.

2.2.3.1 Deployment of Unidirectional MkrSky

It follows the same process of EOA deployment and governance `Spell` initialization as the Legacy MkrSky.

In the initialization library (`updateMkrSky()`):

- The unidirectional MkrSky's immutables mkr, sky and rate are checked to be equivalent to the Legacy MkrSky's.
- The `skyToMkr()` conversion path in the Legacy MkrSky is blocked by removing its `wards` role on `MkrAuthority`.
- Sufficient Sky (`MKR.totalSupply() * rate`) is pre-minted to the unidirectional MkrSky contract.
- The chainlog is updated, the "MKR_SKY" entry is updated to the unidirectional MkrSky contract, and the Legacy MkrSky takes the "MKR_SKY_LEGACY" entry.

In Version 8, functions to disable the Legacy MkrSky were added:

- `disableOldConverterMkrSky()`: revoke the wards role of Legacy MkrSky on SKY, hence `mkrToSky()` will revert.
- `burnExtraSky()`: since some MKR may be converted to SKY through Legacy MkrSky when both converters exist, the excess SKY pre-minted to the unidirectional MkrSky will be burned according to the current MKR total supply.

2.2.4 Changelog

In Version 2 the deployment scripts were added to scope.

In Version 4:

- Functions `increaseAllowance` and `decreaseAllowance` have been removed. Only functions `approve` and `permit` remain to modify allowances.
- Permit functionality: Now, when validating a signature with a contract, if there is no code deployed at the given address, the transaction will revert with a clear error message. Previously the transaction would just revert.

In Version 5, NGT (New Governance Token) has been renamed to SKY.

In Version 6, the SKY token contract will also be used as the L2 token for SKY. Note that the deployment script for the L2 token is slightly different from the L1 token:

- The `MkrSky` converter will not be deployed on L2.
- Note that in contrast to the L1 token, no init script is provided. Namely, that is due to the L2 bridge spells performing `rely` on the tokens (bridge is minter, see for example OP Token Bridge).

In Version 7, the unidirectional converter MkrSky was implemented. The deployment scripts were also adjusted to deploy the new converter.

In Version 8, functions to disable the Legacy MkrSky were added to the SkyInit library.

2.3 Trust Model

Wards of SKY are the only privileged role. Besides managing the addition and removal of wards, their privileges also includes:

- Directly minting new SKY tokens.
- Indirectly diluting MKR by minting new SKY tokens.

It is expected that MkrSky is the only ward of SKY or at minimum the sole ward minting SKY tokens. Otherwise, MKR tokens and MakerDAO can be manipulated if additional SKY tokens are minted and converted to MKR by malicious wards.

The contracts are deployed without proxy.

Sky

Wards are the only privileged roles; fully trusted; otherwise, a malicious ward can:

- Add or remove other wards.
- Mint SKY tokens and indirectly dilute MKR with the converter.

On L1, the Wards of Sky are assumed to be the governance pause proxy and the MkrSky converters. The wards of Sky on L2 is expected to be the L2 Governance Relay and the L2 Token Bridge.

(Legacy) Bidirectional MkrSky

It operates trustlessly without any privileged roles.

Unidirectional MkrSky

Wards are the only privileged roles; fully trusted; otherwise, a malicious ward can:

- Add or remove other wards.
- Set conversion `fee`.
- Steal accumulated fees.
- Block the conversion by burning all the SKY tokens held by the contract.

Deployments

Deployers are supposed to deploy the contracts as specified by the reviewed script. Deployers are, however, EOAs that could perform unlawful actions besides simple deployment, such as changing the settings of the system or granting themselves special privileges. It is important that after deployment, concerned parties thoroughly check the state of the deployed contracts to ensure that no unexpected action has been taken on them during deployment.

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

- Rate Is Not Validated Code Corrected

6.1 Rate Is Not Validated

Correctness Low Version 2 Code Corrected

CS-NGT-001

`NgtInit.init()` does not validate the correct setting of the `rate` parameter in `MkrNgt`. This parameter could be set to an arbitrary value by the contract deployer.

Code corrected:

Since Version 3, the `rate` is checked for correctness in the initialization script.

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 2

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. While some variables can be checked upon initialization through the `PauseProxy`, some things have to be checked beforehand.

We therefore assume that all mappings in the deployed contracts are checked for any unwanted entries (by verifying the bytecode of the contract and then looking at the emitted events). This is especially crucial for `wards` mappings.

7.2 MkrSky Converter Can Be Paused if MKR Is Paused

Note Version 1

The MkrSky converter itself is permissionless, however, if MKR token is paused, the converter will be indirectly paused as `mint()` and `burn()` will revert on MKR.

7.3 Unidirectional MkrSky Conversion Fee May Be Bypassed

Note Version 7

The conversion fee can be enabled in the unidirectional MkrSky contract. The fee may be bypassed in the following scenarios:

1. If the legacy bidirectional MkrSky contract is not disabled, users may keep using the legacy contract for conversions without a fee.
2. If the input `mkrAmt` is low enough, the fee may be rounded down to zero, though this should be non-profitable in practice due to the gas cost of the transaction.