PUBLIC

Code Assessment

of the USDD on Ethereum and BSC Smart Contracts

October 24, 2025

Produced for

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Decentralized USD team,

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

Decentralized USD implements USDD on Ethereum and BNB Smart Chain (BSC). USDD is a fork of the legacy MakerDAO protocol, enabling users to mint USDD, a dollar pegged stablecoin, against various collaterals.

Our review focuses on the changes introduced to deploy USDD on Ethereum and BSC. ChainSecurity has previously reviewed the system which is deployed on Tron (USDDv2 and PSM).

The critical subjects covered in this review are integrations with other tokens, security of the emergency shutdown, and correctness of the deficit auction mechanism (Flopper). Security regarding all aforementioned topics is high.

The general subjects covered in this review are events handling, and documentation. Security regarding all aforementioned topics 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 `usddv2-contracts` and `psm` repositories based on the documentation files.

Our review focuses exclusively on the changes with respect to our previously-audited version of the projects. In particular, the scope of this review is limited to:

- the changes introduced since commit `adb799135095620b2909abe449af174102bf38e2` for the USDDv2 Core repository.
- the changes introduced since commit `c368445876334664d6e58b7e186c78162271ff93` for the PSM repository.

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

USDDv2 Core

V	Date	Commit Hash	Note
1	19 Sep 2025	6166a18fab3abcb1f8395707f613923d0278f534	Initial Version
2	08 Oct 2025	8c50808d3846f419dca098de1d53d9cf70f5729b	After Intermediate Report

PSM

V	Date	Commit Hash	Note
1	19 Sep 2025	b2fcd82ddbe0d37c305cea4e9c2a8314b6874768	Initial Version
2	08 Oct 2025	78688c9c0ae1e0bc4efb90060903ce32ce428d2e	After Intermediate Report

For the solidity smart contracts, the compiler versions `0.6.12` was chosen, except for `SavingUsdd.sol`, which has version `0.8.17`.

The following files are in the scope of the USDDv2 repository:

- src/join-5-auth.sol
- src/join-7-auth.sol
- src/join-auth.sol

The following files are in the scope of the PSM repository:

- src/manager/DssProxyActions.sol
- src/proxy/proxy.sol
- src/dsr/pot.sol
- src/dsr/SavingsUsdd.sol
- src/dss/flop.sol

- src/esm/end.sol
- src/esm/ESM.sol

2.1.1 Excluded from scope

Any other file not explicitly mentioned in the scope section is considered out of scope. In particular, the tests and external dependencies are not part of this audit.

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.

Decentralized USD offers USDD V2, a fork of the legacy MakerDAO protocol (now renamed Sky). It enables users to mint USDD stablecoins using various collaterals.

This version of the project will be deployed on the Ethereum chain and was reviewed in comparison to its previous version, meant to be deployed on the Tron chain. Therefore, we defer to our previous audits (USDDv2 and PSM) for a comprehensive system overview and focus on the differences.

2.2.1 USDD Core Context

- In the USDD system, the core accounting is handled by the **Vat** module, which tracks collateral locked, debt issued, and internal USDD balances.
- Collateral types (“ilks”) are enabled by governance and plugged into the system via **Join adapters**, which bridge between external token representations and the Vat's internal accounting.
- The **Pot** module allows depositors to lock USDD and earn yield.
- The **Flopper** module manages system deficit: it auctions a configurable token (Gem) for USDD to cover debt.
- The **End / Emergency Shutdown (ESM)** modules coordinate a graceful or forced shut-down, freezing new activity, fixing internal prices, and enabling users to redeem their positions (vaults, USDD).

2.2.2 Changes to the previous version

2.2.2.1 pot.sol (new)

This contract is a fork of MakerDAO's pot.sol. The only difference to the original contract is the addition and emission of events.

2.2.2.2 SavingsUsdd.sol (new)

This contract forks SavingsDAI.sol. References to DAI have been renamed USDD. It uses `pot.sol` to offer a tokenised version of the savings USDD (analogous to sDAI in Sky's original design). Depositors receive shares (sUSDD) that accrue yield.

2.2.2.3 join-5-auth.sol

This contract is identical to MakerDAO's join-5.sol. An authenticated join that is designed to handle ERC20 tokens with fewer than 18 decimals. It is used by the PSM to handle USDC on Ethereum, which has 6 decimals: USDC on Ethereum is found at:

- USDC: 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48.

2.2.2.4 join-7-auth.sol (new)

This contract is an authenticated fork of MakerDAO's join-7.sol. An authenticated join meant to be used with the Tether Token (USDT) on Ethereum. USDT is not ERC20 compliant, as its `transfer()` and `transferFrom()` functions return no value. It allows upgrades and potential fees on transfer. In case of upgrade the new implementation has to be whitelisted by the join's administrator. USDT on Ethereum is found at:

- USDT: 0xdAC17F958D2ee523a2206206994597C13D831ec7.

2.2.2.5 src/join-auth.sol (new)

This contract is an authenticated join forked from MakerDAO's join-auth.sol. It supports standard ERC20 tokens with 18 decimals. It is used by the PSM on BSC to handle BNB-USDC and BNB-USDT.

Their addresses are:

- BSC-USDT: 0x55d398326f99059ff775485246999027b3197955.
- BSC-USDC: 0x8ac76a51cc950d9822d68b83fe1ad97b32cd580d.

2.2.2.6 DssProxyActions.sol

The contract was changed from the previous USDD version (Tron) in the following way:

- The contract works with ETH amounts, as opposed to TRX amounts. Therefore, the conversions to 18 decimals were removed (TRX has 6 decimals as opposed to ETH 18).

2.2.2.7 flop.sol

The USDD/MakerDAO's contract has been changed in the following ways:

- The (governance) gem sold in the auction can be changed by the governance.
- If the gem is set to zero, `deal()`, `deposit()`, `withdraw()` and `kick()` are disabled.

2.2.2.8 end.sol

The previous USDD contract has been changed in the following way:

- The pot was added to the shutdown `cage()` function.

2.2.2.9 ESM

The previous USDD contract has been changed in the following way:

- The (governance) gem used to vote on the protocol shutdown can be changed by the governance.

2.3 Trust Model

The admin oversees the entire USDD system and hence is fully trusted. The admin is able to trigger the execution of any privileged functions in the system through the `DSPauseProxy` (i.e. with `customExec()` and `execSpell()` defined in `GovActionsProxy`) and access the funds of users.

The PSM allows the minting of USDD at a 1:1 ratio to USDC and USDT on Ethereum, and BSC-USDC and BSC-USDT on BSC. USDD is therefore pegged to the token of these with the lowest value.

For the complete list of roles in USDD V2, please see our reports (USDDv2 and PSM).

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

- Missing Events Code Corrected

Informational Findings	1

- Inaccurate Natspecs Code Corrected

6.1 Missing Events

Correctness Low Version 1 Code Corrected

CS-USDD-ETH-004

When managing the ward roles, the contract typically emits the `Rely` and `Deny` events. However, the following contracts do not emit such events:

- in `flop.sol` the `rely()` and `deny()` functions do not emit an event. Additionally, the constructor of the contract does not emit a `Rely` event.
- in `pot.sol`, the constructor does not emit a `Rely` event.

Acknowledged: The `Rely` and `Deny` events have been added.

6.2 Inaccurate Natspecs

Informational Version 1 Code Corrected

CS-USDD-ETH-003

A few comments in the codebase are inaccurate.

1. `flop.sol` includes the following comment at line 38:

```
This thing creates gems on demand in return for usdd.
```

The version of Flopper intended to be deployed on Ethereum and BSC has been modified to transfer the gem awarded from the auction from its own balance, instead of minting it as in the original Maker DAO version. The comment is therefore inaccurate.

2. `join-auth.sol` contains the following comment in the first line:

```
join.sol -- Basic token adapters
```

The file name and its description are inaccurate.

Code corrected: Decentralized USD has corrected the inaccurate comments.

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 Burn Function of ESM Might Not Work

Informational Version 1 Acknowledged

CS-USDD-ETH-001

The `burn()` function of ESM calls `gem.burn()` to dispose of permanently locked tokens. If the `gem` configured in the ESM does not expose a public `burn()` function, or if its `burn()` function is permissioned, then `ESM.burn()` will always revert.

Anyway, `ESM.burn()` is not necessary to ensure the correct functioning of the ESM contract.

Acknowledged: Decentralized USD has acknowledged the issue and decided to leave the code unchanged.

7.2 Event in Pot Drip Has Redundant Field `rho`

Informational Version 1 Acknowledged

CS-USDD-ETH-002

Event `Drip(uint256 chi, uint256 rho)` is emitted at every call of `drip()`. The event's field `rho` is always equal to the current block timestamp, which is anyway recorded in the emitted event, making the `rho` field redundant.

The event is also emitted if `drip()` has already been called at the current timestamp, even if calls after the first one at a given timestamp do not change the contract state.

Acknowledged: Decentralized USD has acknowledged the issue and decided to leave the code unchanged.

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 ESM Min Parameter Must Always Be Set

Note Version 1

When deploying USDD on Ethereum and BSC, it is essential that the `min` parameter of the ESM contract is set to a non-zero amount, even if `gem` is initially configured as address zero. Setting `min` to zero will make the `fire()` and `denyProxy()` functions of ESM immediately callable by anyone, leading to a shutdown of the protocol.

8.2 Gem Change After ESM.join

Note Version 1

The `Sum` variable inside `ESM.sol` counts the amount of governance tokens locked inside the contract through `join()`. When `Sum > min` the protocol shutdown can be fired. It is important to note that if any amount of tokens is locked inside the contract, a change in the gem token could lead to an inconsistent comparison against the `min` threshold in case:

- the threshold is not adjusted accordingly
- the new gem has a different number of decimals

8.3 Gem Change During Flop Auction

Note Version 1

The contract `flop.sol` allows the governance to change the gem token being sold in the auction. It is important to note that changing the gem during an auction leads to incorrect behaviors. For instance, a bidder could win an auction by bidding an amount of gem worth more than the previous bidder.