PUBLIC

Code Assessment

# of the USDD V2 Smart Contracts

January 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 V2 according to Scope to support you in forming an opinion on their security risks.

Decentralized USD implements USDD V2, a fork of the MakerDAO Protocol (now Sky) on the Tron blockchain. It enables users to mint USDD stablecoin using various collaterals.

Our review focuses exclusively on code security issues introduced by the changes against the forked codebase. The review does not cover any economic risks. Any errors made by privileged users of the system, including those due to misunderstanding the intricacies and caveats of the forked code base, are out of scope.

The most critical subjects covered in our audit are asset solvency, functional correctness, and access control. Security regarding asset solvency is high.

In the latest version of the codebase:

- Functional correctness has been improved since Incorrect Bar Mechanism in Median and Missing Decimal Upscaling in TRXJoin were resolved.
- Access control has been improved since Access to DSPauseProxy Is Not Restricted to DSPause and GovActionsProxy Will Lose Control Over DSPause if It Changes Delay to Nonzero were resolved.
- In addition, Denial of Service in Median Due to Revert on Invalid Price was partially corrected and the risk of Governance Delay is Currently Disabled was accepted. Hence active monitoring is required to ensure the oracle and governance work correctly.

The general subjects covered are event handling, specifications, and precision of arithmetic operations, which are further improvable, see Events Are Improvable , Incorrect Specifications, and Loss of Precision in Price Calculation Due to Scaling Logic.

In summary, we find that the codebase provides a satisfactory level of security.

Continuing to allocate sufficient time for more extensive internal QA would further increase the security level of the codebase.

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	2
• Code Corrected	1
• Code Partially Corrected	1
Medium-Severity Findings	4
• Code Corrected	3
• Risk Accepted	1
Low-Severity Findings	6
• Code Corrected	1
• Code Partially Corrected	1
• Risk Accepted	4

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 USDD V2 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	16 Dec 2024	25cfa536c70b8b4df3563d4b1f41936c9c7cc291	Initial Version
2	24 Jan 2025	adb799135095620b2909abe449af174102bf38e2	After Intermediate Report

For the solidity smart contracts, the compiler version `0.6.12` was chosen. Except for `Multicall.sol`, which uses the compiler version `0.8.18`.

The following files were in the scope of this review:

```
src/
    base/
        Multicall.sol
    dss/
        dog.sol
        jug.sol
        flap.sol
        abaci.sol
        spot.sol
        join.sol
        clip.sol
        vat.sol
        usdd.sol
        vow.sol
        flop.sol
    dss-deploy/
        DssDeploy.sol
    esm/
        ESM.sol
        end.sol
    gemjoins/
        join-6.sol
        join-7.sol
        join-trx.sol
    gov/
        DSPause.sol
        DSPauseProxy.sol
        DSRoles.sol
        GovActions.sol
        GovActionsProxy.sol
        auth.sol
    manager/
        DssCdpManager.sol
        DssProxyActions.sol
        GetCdps.sol
    osm/
        oracles/
                BaseValue.sol
                OracleValue.sol
                TrxOracleValue.sol
        OsmMom.sol
        median.sol
        medians.sol
        osm.sol
        value.sol
        values.sol
    proxy/
        proxy.sol
        ProxyRegistry.sol
```

The main focus of the review are the changes introduced to the original MakerDAO smart contracts.

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.

The **economic model** of USDD V2 is out of scope.

In addition, USDD V2 relies on proper configurations and active monitoring by its governance. There are many ways in which the governance can cause the system to break or lead to losses for its users. Listing those configurations and governance actions is out of scope for this review.

2.2 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

At the end of this report section, we have added subsections for each of the changes according 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.

Decentralized USD offers USDD V2, a fork of the MakerDAO Protocol (now Sky) on the Tron blockchain. It enables users to mint USDD stablecoin using various collaterals.

USDD V2 preserves most functionalities of the MakerDAO Protocol with various changes. Particularly it forks and modifies the following components:

- dss: The core system of MakerDAO, including the collateral and debt management.
- esm: The Emergency Shutdown Module.
- gemjoins: The join adapters for different collateral types.
- manager: The wrapper for the DSS that improves the users interactions with their CDPs (Collateralized Debt Positions) and where CDPs can be transferred between owners.

- osm: The modified Oracle Security Module that provides the price feeds to the system.
- gov: The modified Governance Module that oversees the USDD V2 system.
- proxy: Users deploy their own `DSProxy` for interacting with the system via delegate calls.

The most important changes are described in the following sections, except for changes to events, which have been updated in most contracts. For more details on the MakerDAO system itself, please refer to the Maker Technical Documentation.

2.2.1 DSS

DSS is the core component of the stablecoin system.

Contracts with no functionality changes in USDD V2

1. Vat: the vault engine that keeps the accounting of CDPs, USDD, and collaterals with the rules by which they can be manipulated.
2. Vow: the debt settlement module that manages the system surplus and bad debt.
3. Usdd: the Dai equivalent and TRC-20 compatible stablecoin contract.
4. Spot: the liaison between the oracles and the core contracts.
5. Jug: the stability fee module that calculates the stability fee for each collateral type.
6. Abaci: the helper contract that computes the price decrease with different models.
7. Dog: the Liquidation 2.0 module where liquidation of the unsafe CDPs can be started.
8. Clip: the liquidation contract per ilk (collateral) using Dutch auctions.

Contracts modified in USDD V2

1. Join: no changes for adapters of gem tokens with 18 decimals (`GemJoin`) and Usdd stablecoin (`UsddJoin`). `TRXJoin` for the native token TRX (with 6 decimals) is added.
2. Flop: the debt auction module. In the MakerDAO system, it is used to cover debt by minting new governance tokens (MKR) and selling them for a fixed amount of DAI. In USDD V2, the Flop module no longer mints governance tokens. Instead, it sells an existing gem token (JST) that must be pre-deposited into the contract. Note that there is no guarantee that there is always sufficient JST in the contract to cover the debt during the auctions. The following functions are added:
   - `deposit()`: anyone can deposit gem tokens by a `transferFrom()`.
   - `setOperator()`: an operator can be configured by the wards.
   - `withdraw()`: the operator can withdraw the gem tokens (JST) to a designated address.
3. Flap: the surplus auction module that sells the surplus of the system for a gem token. In MakerDAO system, it is used to sell surplus DAI for MKR, which is then burned. In USDD V2, Flap has been fully repurposed. Instead of auctioning surplus USDD to acquire and burn a token, it directly exits USDD to a specified receiver during each `kick()` call.

Contracts removed in USDD V2

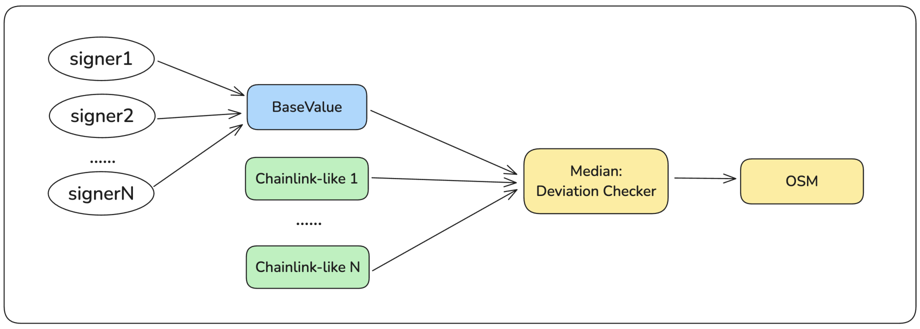
1. Cat: the legacy liquidation module is not used in USDD V2.
2. Flip: the legacy liquidation module is not used in USDD V2.
3. Pot: the savings token is not enabled in USDD V2.
4. Cure: the cure is removed since USDD V2 does not use the teleport module to bridge USDD cross-chain.

2.2.2 OSM

In USDD V2, the Oracle system differs from Maker's design.

The primary price feeds consist of three types of oracles, which will be aggregated into the USDD V2 Median contract, and get delayed for 1 hour in the Osm contract.

- BaseValue: a modified version of the Maker Median contract. The Maker Median contract maintains a whitelist of addresses that are authorized to post price updates. For each update the median is computed and updates the stored value. Based on Median, `BaseValue` adds a `ttl` (time to live) variable which is checked in each `peek()` to indicate the price freshness. `ttl` is configurable by the wards with `setTTL()` and is initialized as 1 day. In addition, `peek()` and `read()` now returns bytes32 instead of `uint256`.
- OracleValue: a wrapper over a Chainlink interface compliant oracle which queries the last round data in each `read()` and `peek()`. A freshness check with `ttl` is also added.
- TrxOracleValue: this is an oracle designed for assets without a direct price feed in USD. It wraps two Chainlink interface compliant oracles: one for the pair Asset/TRX and the other for TRX/USD. In each `read()` and `peek()` it queries both prices and converts to the price of Asset/USD. A freshness check with `ttl` is also added.



In USDD V2, Median is a different contract to Maker's Median though sharing a similar interface. USDD V2 Median, controlled by the wards, will aggregate the results from the primary price feeds configured by `lift()` and `drop()`. Among the authorized price feeds a main oracle can be set. `poke()` will only succeed and return the main oracle price if there are at least `bar` price feeds within the max deviation of the main oracle.

There is no functionality change in the Osm, OsmMom, and DSValue contracts.

2.2.3 Gemjoins

The following join adapters for specific gem tokens are unchanged:

- GemJoin6: join for token with a proxy and implementation contract like tUSD.
- GemJoin7: join for an upgradable token like USDT which doesn't return bool on transfers and may charge fees.

GemJoinTrx newly added for the wrapped version of Tron’s native token TRX (with 6 decimals), is effectively a copy of GemJoin5 — the MakerDAO join adapter for tokens with fewer than 18 decimals (e.g., USDC).

2.2.4 ESM

The ESM (Emergency Shutdown Module) stays unchanged. The End has been changed to remove the logic tied to the legacy liquidation module and the unused Pot and Cure.

In addition, USDD V2 will use JST as the gem token, which can be burnt to trigger the emergency shutdown if the sum passes the threshold.

2.2.5 Proxy

There is no functionality changes in the `ProxyRegistry` contracts, which can deploy new `DSProxy` with `DSProxyFactory` for users and keep track of the owner to its `DSProxy` address.

The `DSProxyFactory` is linked to a `DSProxyCache` that caches the address of deployed byte code.

Users can execute customized code in the context of their `DSProxy` with `delegatecalls`. The `execute()` logic is modified in USDD V2, instead of returning the whole return data from the `delegatecall`, only the first 32 bytes are returned.

2.2.6 Manager

The `DssCdpManager` and `DssProxyActions` implement convenient interfaces for users / personal `DSProxies` to interact with the core system.

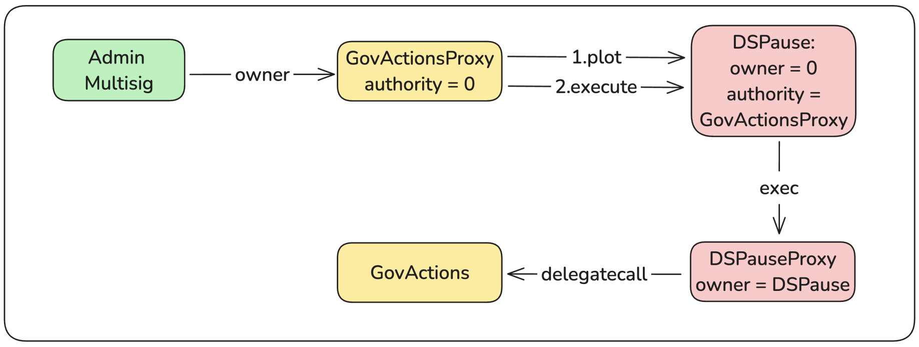
- DssCdpManager: no functionality changes in USDD V2 except a new field `ilk` is added to the event `NewCdp`.
- GetCdps: no functionality changes in USDD V2.
- DssProxyActions: the functionalities related to GNT token and Pot and are removed, and the support of Ether is changed to TRX with 6 decimals. Other functionalities stay the same.

2.2.7 Governance

USDD V2 features a different governance compared to MakerDAO. Instead of being governed by the holder of a governance token, an admin multisig oversees the system with the following contracts:

There are no functionalities change in USDD V2 for the following contracts:

- DSAuth: features `auth` access control with owner and authority.
- DSPause: inherits `DSAuth`, define the process of scheduling and executing a governance spell with delays.
- DSRoles: features granular role-based access controls for contracts and selectors.


Admin Multisig
owner
GovActionsProxy authority = 0
1.plot
2.execute
DSPause: owner = 0 authority = GovActionsProxy
exec
GovActions
delegatecall
DSPauseProxy owner = DSPause

The `DSPauseProxy` is the contract that bear the privileges over the system contracts and execute the governance spells. In MakerDAO it is can only be called by its owner: `DSPause`, however, in USDD V2, it is modified and inherits `DSAuth`. Hence, executions can be triggered (`exec()`) by any authed parties (if an `authority` contract is specified) with enabled privileges to call `DSPauseProxy`.

`GovActions` is a set of predefined governance helper functions that can be executed by the `DSPauseProxy`. A new function `customExec()` is added to allow the execution of arbitrary calls.

`GovActionsProxy` is expected to be the authority of `DSPause` and owned by the admin multisig. A set of helper functions are defined for admin to schedule (`plot()`) and execute (`exec()`) functions through the `DSPause` and `DSPauseProxy` in one transaction without delays. `customExec()` and `execSpell()` are also provided to allow the admin to trigger arbitrary execution from the

DSPauseProxy without delays. Note that GovActionsProxy will only work if delay of DSPause is set to 0.

2.2.8 TronMulticall

A TronMulticall contract is provided, which provides view functions to retrieve block context (i.e. chainid, timestamp, blockhash), native token balance, TRC-10 balance, and if an address is a contract (isContract). This is a helper contract that can query on-chain data in batch and hence reduce the number of off-chain calls. Note there are some differences in the opcode semantics between TVM and EVM.

2.2.9 Changes in Version 2

The following changes were made in this version:

- An auth-ed function pokeOracle has been added to the Median contract, where an authorized oracle's valid price can be directly poked into Median, bypassing the main oracle and price deviation check of poke() in emergency.
- DSAuth has been removed from DSPauseProxy, and DSPauseProxy now can only be called by its owner (expected DSPause).
- Functions schedule (restricted to wards) and cast have been added to the GovActionsProxy that allow the admin to schedule a customExec call respecting the governance delay, and execute the call once the delay has passed.

2.2.10 Roles & Trust Model

The USDD V2 system features the following privileged roles:

- The admin oversees the entire USDD V2 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.
- If the authority of GovActionsProxy is set, any roles that can pass the authority.canCall() are fully trusted. Otherwise, they may abuse their specific privileges within the system for malicious manipulations.
- The operator of Flop is fully trusted. Otherwise, it can withdraw all the gem balance in the contract, effectively blocking all the auctions and the winning bidder will not be able to get the gem tokens.
- The receiver of Flap will receive the system surplus in each kick() and hence is trusted. Otherwise, the system surplus is forfeited.
- The majority of the Oracles (Value contracts) signers are trusted. Otherwise, they can collude and manipulate the price within the allowed max deviation and bar to arbitrage the users.
- The wards of the Median oracle are fully trusted. Since **Version 2**, they can bypass the main oracle and the deviation check with pokeOracle(), a function that pokes a price from a single, registered and chosen oracle.

The proper deployment and operation of various components in the USDD V2 system depend on the contracts' liaison, the privileges' assignment, and the correct configuration of different parameters. It is assumed that the admin has deployed and configured these correctly, for instance:

- The oracles modules are deployed and connected correctly, the primary oracles are wired to the Median contract for deviation check and eventually linked to the OSM.
- The auth-ed address on OsmMom is correctly configured and trusted. Otherwise, it can trigger the stop() which blocks price updates to Osm contracts.

- The implementation and risk of each ilk are properly analyzed before enabled in the system. Otherwise, the system and its users are exposed to the risks of such ilks.
- A suitable join contract will be selected and deployed from the available options to ensure compatibility with the ilk it is associated with.
- In Vow, `bump` should be a multiple of `RAY`. Otherwise, flap will have dust USDD amount that cannot be exited due to rounding errors.
- In Flapper, the hop is set to a reasonable value. In case the hop is set too large, the computation of `zzz + hop` may overflow and result in a small value compared to `block.timestamp`, which effectively allows multiple kicks without delays.
- In Spot, `par` is set to `ONE` (ref per USDD), hence the price obtained from `pip.peek()` should be a price denominated in an asset that has a 1:1 conversion rate to USDD. Otherwise, the price conversion will be incorrect.
- The stability fee, liquidation threshold / penalty, debt limits, etc. are configured properly. **Bad parameters may incur loss to the users or the depegging of USDD**.
- ESM should have privileges to call burn function of the gem token. Otherwise, the gem tokens accumulated in the contracts cannot be burnt.
- The threshold to trigger emergency shutdown is set properly according to JST. Otherwise, it may be impossible or economically-profitable to maliciously trigger the emergency shutdown. In addition, the minters of JST are fully trusted, otherwise they can mint JST and trigger the shutdown easily.
- Each governance spells are carefully inspected and simulated. Otherwise, an unintended mistake in the spell may be executed without any chance for termination due to the currently disabled governance delay in `DSPause`.

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 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
- Design: Architectural shortcomings and design inefficiencies
- 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	1

- Denial of Service in Median Due to Revert on Invalid Price Code Partially Corrected Risk Accepted

Medium-Severity Findings	1

- Governance Delay Is Currently Disabled Risk Accepted

Low-Severity Findings	5

- Authed PokeOracle Does Not Check Prive Validity Risk Accepted
- Events Are Improvable Code Partially Corrected
- Loss of Precision in Price Calculation Due to Scaling Logic Risk Accepted
- Signed Message May Be Interchangeable Between Tron and Ethereum Risk Accepted
- Unsafe Use of Assembly Risk Accepted

5.1 Denial of Service in `Median` Due to Revert on Invalid Price

Design High Version 1 Code Partially Corrected Risk Accepted

CS-USDD-V2-001

In the `poke` function of the `Median` contract, if a single oracle returns an invalid price or exceeds the allowed deviation, the function reverts. In addition, the `BaseValue` and `TrxOracleValue` contract can revert in `peek()` if the any require statement is not satisfied (i.e. checks of freshness, roundId...). This can create a Denial of Service vulnerability as the presence of any faulty oracle prevents the price from being updated even if more than `bar` oracles returned valid prices within the max deviation. Hence if any of the oracle is compromised, it can block the price update of the `Median` contract, which requires wards privilege to recover the contract by dropping the malicious oracle.

Code partially corrected and risk accepted:

The code has been partially corrected to avoid reverts in `Median.poke()`: it was modified to skip invalid or deviating oracle responses instead of reverting

However, the `BaseValue` and `TrxOracleValue` contract can still revert in `peek()` due to the round data checks, hence DoS the `Median.poke()`.

Decentralized USD has accepted the risks with the following response:

```
We will monitor the oracle price and take actions when needed and we
plan to fix it in the next version.
```

5.2 Governance Delay Is Currently Disabled

Security **Medium** **Version 1** **Risk Accepted**

CS-USDD-V2-004

USDD V2 is under the control of the admin multisig, which can use `GovActionsProxy` to schedule (`plot()`) an execution with `eta=now` and execute (`exec()`) it immediately assuming the governance delay defined in `DSPause` is set to 0.

This effectively disables the governance delay and allows instant execution of privileged actions. For instance, in the event enough signers of the multisig are compromised, they could change the owner of the system instantly, without any delay. The remaining honest signers of the multisig will not be given any time window to react and prevent the malicious execution. The risk is escalated with the `customExec()` defined in `GovActions` that can allows arbitrary calls from the `DSPauseProxy`.

Risk accepted:

Decentralized USD has acknowledged that the `DSPause` governance delay is currently disabled with the following response:

```
USDD is currently controlled by the admin multisig. We will actively develop the
governance module to transition control to governance token holders.
```

5.3 Authed PokeOracle Does Not Check Prive Validity

Security **Low** **Version 2** **Risk Accepted**

CS-USDD-V2-017

In `Median`, an auth-ed function `PokeOracle` was introduced in **Version 2** that enables the wards to update the current price (`val` and `age`) of Median from an authorized oracle and bypass the main oracle price deviation check.

However, the validity flag from calling `peek()` is not checked, hence an invalid price may be poked into Median.

Risk accepted:

Decentralized USD has accepted this risk with the response:

```
This is an adjustment to mitigate price anomalies in the oracle feed. We will
actively monitor the oracle price and will improve that in a future version.
```

5.4 Events Are Improvable

Design Low Version 1 Code Partially Corrected

CS-USDD-V2-007

The events handling in following cases can be improved:

1. In Vat, events `Hope()` and `Nope()` are added. Both events only contain the target `usr` address, hence the events do not reflect who is granting or revoking the privilege to `usr`.
2. In Jug, events `Hope()` and `Nope()` are added, however, never used. In contrast, there is no events for `file()` and `init()`.
3. In Vow, there is no events for `heal()` and `kiss()` while some events are added for other state-modifying functions.
4. In DSPause, there is no event for `setDelay()` while some events are added for other state-modifying functions.

Code Partially Corrected:

Decentralized USD has acknowledged this issue:

1. No changes.
2. Events were added to the `init()` and to all `file()` functions except the one to set `vow`.
3. No changes.
4. No changes.

5.5 Loss of Precision in Price Calculation Due to Scaling Logic

Correctness Low Version 1 Risk Accepted

CS-USDD-V2-008

In the `peek` function of the `TrxOracleValue` contract, precision is lost when computing `usdPrice`. If either `answer` or `trxUsdPrice` has more than 18 decimals, the values are truncated, leading to a loss of precision. This truncation is compounded when the two values are multiplied together, potentially resulting in a significant deviation in the final `usdPrice`.

```
// Scale both prices to 18 decimals
uint256 scaledPrice = scalePrice(uint256(answer), oracle.decimals());
uint256 scaledTrxPrice = scalePrice(uint256(trxUsdPrice), trxOracle.decimals());

// Calculate final USD price: (Asset/TRX) * (TRX/USD)
uint256 usdPrice = (scaledPrice * scaledTrxPrice) / 1e18;
```

In general, performing a division (to scale down) followed by a multiplication compounds precision loss. Performing the entire calculation in full precision first and then scaling the final result provides a more accurate result.

Risk Accepted:

Decentralized USD has accepted the risk of potential precision loss in case tokens are above 18 decimals with following response:

```
Currently, the oracle we will use is the WinkLink oracles (system on the TRON
Mainnet, designed for Chainlink-like oracles) that use 18 decimals.
A modification will be made if an oracle has more than 18 decimals.
```

5.6 Signed Message May Be Interchangeable Between Tron and Ethereum

Security Low Version 1 Risk Accepted

CS-USDD-V2-018

In contracts `BaseValue` and `values`, the function `recover()` will reconstruct the message hash with Ethereum signed message header: `\x19Ethereum Signed Message:\n32`. This will later be used to verify the signer's signature over the price.

In case the signer is signing prices for protocols on both Tron and Ethereum, the signed price for Tron may be replayable on Ethereum and vice versa. This is because both protocols use the same Ethereum signed message header, making the signatures interchangeable.

Risk accepted:

Decentralized USD has accepted the risk of potentially interchangeable signatures with the following response:

```
We will ensure the signers are only used for the TRON network in our SOP solution.
```

5.7 Unsafe Use of Assembly

Security Low Version 1 Risk Accepted

CS-USDD-V2-010

In the `DSProxy` contract, `execute()` initiates a `delegatecall` with assembly where the `retOffset` is 0 and `retSize` is 32 bytes.

```
assembly {
    let succeeded := delegatecall(gas(), _target, add(_data, 0x20), mload(_data), 0, 32)
    response := mload(0)
    switch iszero(succeeded)
    case 1 {
        revert(0, 0)
    }
}
```

If the `delegatecall` returns more than 32 bytes of data, the return data will be truncated to 32 bytes.

After the `delegatecall`, it loads the memory at offset 0 as the response. Since the `delegatecall` does not reset the memory at offset 0 before, if there was any garbage data, this data will be loaded and regarded as return data. Consequently, the calling contract of `execute()` may become vulnerable to the unexpected response data.

In addition, even if there is no garbage data, the calling contract cannot distinguish between the following cases which may lead to unexpected results.

1. The `delegatecall` does not have any return data.
2. The `delegatecall` has return data that is 0.

Risk Accepted:

Decentralized USD has accepted the potential risks with this assembly usage.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0

High-Severity Findings	1

- GovActionsProxy Will Lose Control Over DSPause if It Changes Delay to Nonzero Code Corrected

Medium-Severity Findings	3

- Access to DSPauseProxy Is Not Restricted to DSPause Code Corrected
- Incorrect Bar Mechanism in Median Code Corrected
- Missing Decimal Upscaling in TRXJoin Code Corrected

Low-Severity Findings	1

- Redundant Privilege Granted to USDDJoin in Deployment Code Corrected

6.1 GovActionsProxy Will Lose Control Over DSPause if It Changes Delay to Nonzero

Security High Version 1 Code Corrected

CS-USDD-V2-003

If `GovActionsProxy` is configured with `authority = 0` and owned by the admin multisig, and `DSPause` is configured with `delay = 0`, the admin multisig can immediately execute privileged actions through `GovActionsProxy`:

```
uint eta = now;
PauseLike(pause).plot(usr, tag, fax, eta);
PauseLike(pause).exec(usr, tag, fax, eta);
```

However, if the admin updates the `delay` in `DSPause` to a nonzero value (which will succeed), subsequent `plot()` calls will fail because `eta = now` no longer satisfies `eta >= now + delay`:

```
function plot(address usr, bytes32 tag, bytes memory fax, uint eta)
    public auth
{
    require(eta >= add(now, delay), "ds-pause-delay-not-respected");
    plans[hash(usr, tag, fax, eta)] = true;
    emit Plot(usr, tag, fax, eta);
}
```

Since `GovActionsProxy` has no logic to accommodate a nonzero delay and no other authorized party is able to manage plotting on DSPause — all future governance actions break, effectively causing the admin multisig to lose control of `DSPause` and, by extension, the system.

This also assumes that the owner of `DsPausePauseProxy` is `DsPause` with no authority configured.

Code Corrected:

Two new functions, `schedule` and `cast`, have been introduced to address the issue. The `schedule` function (restricted to wards) allows scheduling a call to `GovActions.customExec` with the `DSPause.delay` which can be used to reset the delay to 0. The `cast` function (permissionless) can trigger the execution of a call with `DSPause.exec()` once `delay` passed (though `DSPause.exec` is also permissionless).

6.2 Access to DSPauseProxy Is Not Restricted to DSPause

Security Medium Version 1 Code Corrected

CS-USDD-V2-002

`DSPauseProxy` bears the ward privileges over the system contracts. In MakerDAO, only `DSPause` can invoke it, preventing unauthorized calls to it. However, in USDD V2, `DSPauseProxy` inherits `DSAuth`. If an authority is configured in `DSAuth`, other addresses may also pass the `auth` modifier if `canCall()` returns true. Consequently, the access to `DSPauseProxy` is no longer strictly restricted to `DSPause`. A malicious actor that passes `canCall()` may abuse the `DSPauseProxy` privileges to attack the system and its users.

Code corrected:

The `DSPauseProxy` implementation was reverted to MakerDAO's version where the access to it is restricted to a single owner.

6.3 Incorrect Bar Mechanism in Median

Correctness Medium Version 1 Code Corrected

CS-USDD-V2-005

The `Median` contract implements a price deviation check in function `poke`: the price of the main oracle will only be valid if there are at least `bar` valid price feeds (including the main oracle itself) that are within the max deviation from the main oracle.

However, when iterating over all other oracles in a loop, there is a require statement that enforces the price to be within the deviation limit or the call will revert. As a result, `validCount` will always equal `orclCount` by the end of the loop, or the execution will have reverted. This renders the `validCount` variable and the `bar` concept meaningless. The `Median` contract will have the same behavior for all `bar` choices from 1 to `orclCount`.

Code corrected:

The `Median.poke` function was modified to skip invalid or deviating oracle responses instead of reverting. Hence the loop will not revert early in case any of the price oracle returns invalid / deviating price.

6.4 Missing Decimal Upscaling in TRXJoin

Correctness | Medium | Version 1 | Code Corrected

CS-USDD-V2-006

TRXJoin is implemented to support the integration with native TRX tokens on Tron. TRX has 6 decimals. TRXJoin directly uses the msg.value for Vat interactions with slip(), whereas Vat expects that the input value is in 18 decimals. Consequently, the internal balance of TRX in Vat will be 10^-12 of the correct value. In case TRX is enabled as an ink with correctly configured parameters and oracles, this will effectively rendering the borrowing power of TRX to be 10^-12 times less.

Code Corrected:

The TRXJoin contract was updated to upscale msg.value to 18 decimals before interacting with the Vat contract.

6.5 Redundant Privilege Granted to USDDJoin in Deployment

Security | Low | Version 1 | Code Corrected

CS-USDD-V2-009

In DssDeploy, deployUsdd() grants USDDJoin the ward role over the Vat. This is a powerful privilege allowing contracts to manipulate the core accounting system and should not be granted to unnecessary addresses. Since USDDJoin has no functionality to invoke any auth-ed functions on the Vat, this privilege is redundant.

Code Corrected:

deployUsdd() has been modified to not grant the unnecessary ward privileges of Vat to USDDJoin.

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 Abstract Contract Can Be Changed to Interface

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

CS-USDD-V2-011

`GovActions` contract implements a set of helper functions for calling different system contracts to set parameters. Three abstract contracts: Setter, EndLike and PauseLike are defined in the same file, which can be changed to interfaces as they do not have any implementation. This will make the code more readable and maintainable.

Acknowledged:

Decentralized USD has acknowledged this issue and decided not to change it.

7.2 Gas Optimizations

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

CS-USDD-V2-012

The gas efficiency could be improved, and the following is a non-exhaustive list of potential optimizations:

- In contracts `GemJoin6`, `GemJoin7` and `GemJoinTrx`, variables `vat`, `ilk`, `gem`, and `dec` will never be changed after deployment, hence can be declared as immutables.
- In the `DSProxy` contract, the `require` statement around the `setCache` and the unconditional `return true` in `setCache` are both unnecessary as the function always returns `true` or reverts.
- In `DssProxyActions` contract, function `freeTRX()` can cache the result of `convertTo18(trxJoin, amt)` instead of computing it twice.

Acknowledged:

Decentralized USD has acknowledged the optimizations and decided not to change it.

7.3 Incorrect Specifications

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

CS-USDD-V2-013

The following specifications are incorrect:

1. In contract End:

    1. Since the liquidation module `cat` and `flip` are removed, the specifications about step *4a* are redundant.
    2. Some specifications uses `dai` instead of `usdd`.
2. In `dss/join`, the specification of `TRXJoin` incorrectly states it is for native Ether.
3. In contract TronMulticall, `getEthBalance()` incorrectly states that it returns the ETH balance of an address.
4. In Median, `wat` has been implemented as a virtual view function explicitly, hence this line of comment is redundant: `bytes32 public constant wat = "ethusd"`.

Specification partially changed:

1. The redundant specifications have been removed and `dai` has been replaced with `usdd`.
2. The specification of `TRXJoin` has been corrected to state that it is for TRX.
3. Acknowledged without changes.
4. Acknowledged without changes.

7.4 Known MakerDAO Issues

Informational **Version 1** Acknowledged

CS-USDD-V2-014

The following are examples of known MakerDAO issues identified in the original system that persist in USDD V2. This list is non-exhaustive and serves as a reminder that the Maker system comes with many potential failures and gotchas that requires monitoring and swift actions:

1. **Double Update at Epoch Boundary**

   In the `OSM` contract, when `poke()` is called near the end of an epoch (i.e., when `era()` is just about to exceed `zzz + hop`), the code updates `zzz` by setting it to `prev(era())`:

   ```
   function prev(uint ts) internal view returns (uint64) {
       require(hop != 0, "OSM/hop-is-zero");
       return uint64(ts - (ts % hop));
   }
   ```

   If the next block crosses the epoch boundary, a second call to `poke()` can immediately succeed, allowing two consecutive updates in quick succession without enforcing the full delay (`hop`).

   To mitigate the manipulation risks, active monitoring of the system and timely actions in emergency are required. For more details please refer to MakerDAO's documentation of OSM Failure Modes.

2. **Median Failure**

   By design, there is currently no way to turn off the oracle (failure or returns false) if all the oracles come together and sign a price of zero. This would result in the price being invalid and would return false on `peek()`, telling us to not trust the value.

3. **Keepers**

Keepers are external actors that play vital roles in MakerDAO system operations. For instance:

- "Auction Keepers" are incentivized to automate the seeking of auction opportunities and bidding. Without them, insolvent positions may not be liquidated in time hence accumulate bad debt to the protocol.
- The End relies on “Cage Keepers” to properly execute the Emergency Shutdown procedure. Without them, undercollateralized vaults may go unprocessed, and Dai (or USDD) holders may be unable to redeem collateral fairly.

4. **System Parameterization**

   The configurations and updates of each ilk should be thoroughly analyzed in advance to mitigate the risks of unpredictable market condition.

Acknowledged:

Decentralized USD has acknowledged this issue with the following response:

```
For Double Update at the Epoch Boundary, we have created SOP solutions to actively
monitor the system, ensuring timely actions are taken in emergencies. These SOP
solutions also monitor Median failures. For keepers, auction monitoring is in place,
and an auction keeper has been set up to ensure liquidations occur on time.
Additionally, careful analysis will be conducted before making any changes to
system configurations.
```

7.5 USDD Hardcodes the Domain Separator

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

CS-USDD-V2-015

The `DOMAIN_SEPARATOR` for computing the permit message digest in USDD contract is hardcoded and initialized in the constructor with the input `chainId`. In the unlikely situation of a chain-split or network hardfork, the `chainId` for one of the forked networks may be changed to prevent network-wise replay attacks. However, the `DOMAIN_SEPARATOR` will be the same on both chains hence signatures will be replayable across both chains.

7.6 Underlying Oracles Have Inconsistent Price Upper Bound

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

CS-USDD-V2-016

For the primary oracles, both `OracleValue` and `TrxOracleValue` support price up to `max(uint256)`, whereas `BaseValue` will truncate the median price from `uint256` to `uint128`. In addition, `OSM` will also truncate the `uint256` price to `max(uint128)`. Consequently, in case the spot price of the underlying asset exceeds `max(uint128)`, the truncated value reported by `BaseValue` will be an outlier (exceed max deviation) compared to the ones reported by `OracleValue` and `TrxOracleValue`. Eventually, the price truncated in `OSM` will be much less than the real spot price. In practice, it is unlikely that an asset has an 18-decimal price that exceeds `max(uint128)`.

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 Implication of the Governance Token

Note Version 1

USDD V2 uses JST as the governance token (gem) in contracts like Flop and ESM, which has different implication compared to MakerDAO:

1. Since Flop cannot mint JST, it is not guaranteed that there is always sufficient JST to cover the system bad debt.
2. JST may have a different tokenomics compared to MKR, hence the governance token holders may have different incentives, and the costs / benefits of governance attacks (i.e. triggering emergency shutdown) may be different. It should be ensured that the threshold is high enough to prevent malicious shutdowns.
3. The existing JST token governs multiple protocols, hence the risks and profits of governance attacks may be increased due to the potential loss on all the governed protocols.

8.2 Potential Phishing Attacks for USDD

Note Version 1

USDD contract is equivalent to DAI, which contains `push()` which is an alias for the `transfer()` function. Compared to a normal TRC-20 token, this may expose additional risks of phishing attacks in case users are unclear with the `push()` functionality. Users of USDD should be aware of the this and avoid potential phishing attacks that targets `push()` to steal users' USDD tokens.