PUBLIC

Code Assessment

of the DSS Vest Smart Contracts

December 12, 2024

Produced for

by

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help MakerDAO with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of DSS Vest according to Scope to support you in forming an opinion on their security risks.

Client implements DssVest, an abstract contract for creating vesting plans, with concrete implementations defining payout methods (mintable, transferable, suckable). In this latest version, DssVestSuckable has been refactored to support USDS.

The most critical subjects covered in our audit are functional correctness and access control. 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	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the DSS Vest 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	9 December 2024	41728f0361418ff4831d0987c2b8aadffbf2f539	Initial Version

For the solidity smart contracts, the compiler version `0.6.12` was chosen.

The following file was in scope:

```
src/DssVest.sol
```

2.1.1 Excluded from scope

Generally, all files not mentioned above are out of scope.

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.

DSS Vest allows to create a vesting plan for recipients. The base contract `DssVest` implements most of the functionality, while the three deriving contracts, `DssVestMintable`, `DssVestSuckable` and `DssVestTransferrable` implement various ways of paying out the vested tokens. In this latest version, `DssVestSuckable` has been refactored to support USDS.

2.3 Vesting Plans

A vesting plan has several parameters:

- `usr`: The recipient address of the tokens supplied by the vesting plan.
- `bgn`: The timestamp marking the beginning of the vesting period.
- `clf`: The timestamp marking the cliff date of the vesting period.
- `fin`: The timestamp marking the end of the vesting period.
- `mgr`: The manager address of the vesting plan.
- `res`: A boolean value marking whether the vesting plan is restricted or not.
- `tot`: The total reward amount for the vesting plan.

- `rxd`: The reward amount that the recipient has received so far.

The reward is distributed linearly over the vesting period. The rate at which it is distributed is determined by `tot / (fin - bgn)`. The rewards start accumulating at the beginning of the vesting period but can only be claimed after the cliff date.

A ward or the manager of the vesting plan may `yank` (terminate) the plan at any time between now (`block.timstamp`) and the end (`fin`) of the vesting plan. Rewards accumulated up to that point remain claimable, but no further rewards will be distributed. Note that if a `yank` occurs before the cliff, no vesting rewards will be distributed.

By default a newly created `Award` (vesting plan) is not restricted. The wards of the `DssVest` or the user can set its `Award` to restricted. If a vesting plan is restricted, the distribution of the rewards can only be triggered by the recipient of the vesting plan. Otherwise, anyone can trigger the distribution of the rewards.

An admin of the system can set a global `cap` for the rate at which vesting plans distribute rewards. A vesting plan with a higher rate than the `cap` cannot be created.

Three ways of rewards payout are derived:

1. `DssVestMintable`: the reward token (gem) will be minted directly to the recipient.
2. `DssVestSuckable`: the reward token comes from the increased `sin` on `MCD_VOW`, and will be exited to the recipient by the `JOIN`.
3. `DssVestTransferrable`: the rewards token will be transferred from a distributor address to the recipient.

2.4 Roles and Trust Model

There are three privileged roles in the system:

1. Ward - A ward is an admin of the system. They may add or remove other wards, create new vesting plans and change the global `cap` for the distribution rate. Lastly, they can restrict, unrestrict or `yank` existing vesting plans. The role is considered trusted by the system and will not be against the interest of its users.
2. Manager - A manager of a vesting plan is determined at its creation. The manager may `yank` the vesting plan, which stops the rewards from accumulating any further. This role is considered trusted by the system that it will not act maliciously.
3. Recipient - The recipient of a vesting plan has certain privileges as well. They may restrict or unrestrict their vesting plan. If the vesting plan is restricted, they are the only one who can trigger distribution of their rewards. Finally, they can `move` the vesting plan, essentially transferring it to another recipient.
4. Everyone else - If a vesting plan is not restricted, anyone can trigger the distribution of rewards (but they still go to the recipient of the vesting plan).

The following assumptions were further made on the concrete vest contracts:

- `DssVestMintable`: The vest should have the minter role on the gem token.
- `DssVestSuckable`: The vest should have `wards` privilege on `Vat` for calling `suck()`.
- `DssVestTransferrable`: The vest should have sufficient gem token allowance from the distributor `czar`.

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 our findings. The findings are split into these different categories:

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 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.

6.1 Redundant Calculation in _yank

Informational **Version 1**

CS-DSSVEST-001

When `_yank` is called, it determines how much reward can still be claimed by the recipient. If the new `_end` of the vesting plan is before the beginning or the cliff date of the vesting plan, the entire reward amount is cancelled. Otherwise, the following calculation is done:

```
awards[_id].tot = toUint128(
    add(
        unpaid(_end, _award.bgn, _award.clf, _award.fin, _award.tot,
        _award.rxd),
        _award.rxd
        )
    );
```

`unpaid` calculates the following result:

```
function unpaid(uint256 _time, uint48 _bgn, uint48 _clf, uint48 _fin,
    uint128 _tot, uint128 _rxd)
    internal pure returns (uint256 amt) {
    amt = _time < _clf ? 0 : sub(accrued(_time, _bgn, _fin, _tot), _rxd);
}
```

As we know that `_end` is after the beginning and cliff date of the vesting plan, this calculation can be simplified. The addition of `_rxd` in `_yank` and subtraction of `_rxd` in `unpaid` cancel out, so the final result is simply: `toUint128( accrued(_end, _bgn, _fin, _tot) )`.

6.2 Unnecessary Calculation in accrued

Informational **Version 1**

CS-DSSVEST-002

In the `accrued` function, unnecessary calculations are performed in the case where `_time == _bgn`. In this case, the result will be `0`. As such, the first if condition could be modified to be `_time <= _bgn` in order to save gas in this case.

```
function accrued(uint256 _time, uint48 _bgn, uint48 _fin, uint128 _tot)
    internal pure returns (uint256 amt) {
    if (_time < _bgn) {
        amt = 0;
    } else if (_time >= _fin) {
        amt = _tot;
    } else {
```

```
            amt = mul(_tot, sub(_time, _bgn)) / sub(_fin, _bgn);
        }
    }
```

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 Function daiJoin May Not Return the Actual DaiJoin

Note Version 1

A view function `daiJoin()` has been added to maintain compatibility with the legacy DssVestSuckable that support DAI only. Now DssVestSuckable is expected to be used with USDS. The `join` address returned may not be the actual DaiJoin, but the respective join adapter for USDS.

While this facilitates compatibility, integrators should be aware of this difference and must be careful that their contract can handle USDS if they interact with this join adapter.

7.2 No Rewards Can Be Claimed if VAT Is Dead

Note Version 1

When rewards of a vesting plan is being claimed in DssVestSuckable, function `pay` will only proceed if `vat.live() == 1`. In case the vat is caged (`cage()`), no tokens can be claimed anymore.