ABDK CONSULTING

SMART CONTRACT AUDIT

MakerDAO

DSSVest

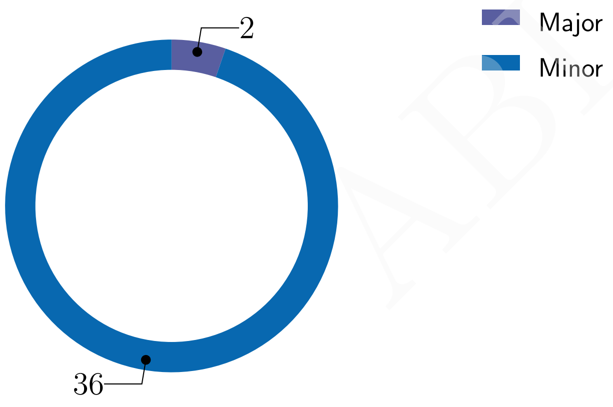
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov
25th July 2021

We've been asked to review the MakerDAO: DSSVest smart contracts given in one file. We have found 2 major and a few minor issues. They were all fixed in subsequent updates.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Info
CVF-2	Minor	Procedural	Info
CVF-3	Minor	Readability	Info
CVF-4	Minor	Readability	Info
CVF-5	Minor	Flaw	Info
CVF-6	Minor	Flaw	Info
CVF-7	Minor	Unclear behavior	Info
CVF-8	Minor	Documentation	Info
CVF-9	Minor	Suboptimal	Info
CVF-10	Minor	Suboptimal	Fixed
CVF-11	Minor	Suboptimal	Fixed
CVF-12	Minor	Bad naming	Fixed
CVF-13	Minor	Suboptimal	Info
CVF-14	Minor	Flaw	Info
CVF-15	Minor	Readability	Info
CVF-16	Minor	Suboptimal	Info
CVF-17	Minor	Flaw	Info
CVF-18	Minor	Suboptimal	Info
CVF-19	Major	Unclear behavior	Fixed
CVF-20	Minor	Readability	Fixed
CVF-21	Minor	Unclear behavior	Info
CVF-22	Minor	Documentation	Fixed
CVF-23	Minor	Flaw	Info
CVF-24	Minor	Procedural	Fixed
CVF-25	Minor	Suboptimal	Fixed
CVF-26	Minor	Suboptimal	Info
CVF-27	Minor	Suboptimal	Fixed

ID	Severity	Category	Status
CVF-28	Minor	Flaw	Fixed
CVF-29	Minor	Procedural	Fixed
CVF-30	Major	Flaw	Fixed
CVF-31	Minor	Documentation	Fixed
CVF-32	Minor	Procedural	Info
CVF-33	Minor	Flaw	Fixed
CVF-34	Minor	Flaw	Info
CVF-35	Minor	Bad datatype	Info
CVF-36	Minor	Readability	Info
CVF-37	Minor	Bad datatype	Info
CVF-38	Minor	Readability	Fixed

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	July 24, 2021	M. Vladimirov	Initial Draft
0.2	July 25, 2021	M. Vladimirov	Minor revision
1.0	July 25, 2021	M. Vladimirov	Release
1.1	August 18, 2021	M. Vladimirov	Add client comment
2.0	August 18, 2021	M. Vladimirov	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

We audited the DssVest.sol file at commit d05700. The fixes were applied in pull request 34.

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.

- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.

- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.

- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** DssVest.sol

Recommendation Should be "Ô.6.0" according to a common best practice, or "Ô.6.12" in case there is something special about the 0.6.12 version.
Client Comment Bytecode and FV testing against 0.6.12 so we can't verify accuracy for other versions.

Listing 1:

```
solidity 0.6.12;
```

3.2 CVF-2

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** DssVest.sol

Recommendation These interfaces and contracts should be moved to separate files named after them. Putting several contracts and interfaces into a single file makes it harder to navigate through the code.
Client Comment They are too simple to clutter the project with more files and this is actually a subjective item. We do consider is clearer this way.

Listing 2:

```
MintLike {

ChainlogLike {

DaiJoinLike {

VatLike {

DssVestMintable is DssVest {

DssVestSuckable is DssVest {
```

3.3 CVF-3

- **Severity** Minor
- **Category** Readability
- **Status** Info
- **Source** DssVest.sol

Recommendation The code could be made more readable by giving descriptive names to the function arguments.
Client Comment We prefer not to add arguments name to the interfaces.

Listing 3:

```
function mint(address, uint256) external;

function getAddress(bytes32) external view returns (address);

function exit(address, uint256) external;

function hope(address) external;
function suck(address, address, uint256) external;
```

3.4 CVF-4

- **Severity** Minor
- **Category** Readability
- **Status** Info
- **Source** DssVest.sol

Recommendation This value could be rendered as "1e18".
Client Comment We use WAD for explicitness.

Listing 4:

```
uint256 internal constant WAD = 10**18;
```

3.5 CVF-5

- **Severity** Minor
- **Category** Flaw
- **Status** Info
- **Source** DssVest.sol

Recommendation This variable should have type "bool" as it acts as a boolean flag.
Client Comment Subjective change. uint256 will be more gas efficient to use.

Listing 5:

```
uint256 internal locked;
```

3.6 CVF-6

- **Severity** Minor
- **Category** Flaw
- **Status** Info
- **Source** DssVest.sol

Recommendation The value type for this mapping should be "bool".
Client Comment Same prev one.

Listing 6:

```
mapping (address => uint256) public wards;
```

3.7 CVF-7

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** DssVest.sol

Description The events are logged even if nothing was actually changed.
Client Comment We prefer not adding logic to emit or not the event when it's not really needed.

Listing 7:

```
function rely(address usr) external auth { wards[usr] = 1; emit
    ↪ Rely(usr); }
function deny(address usr) external auth { wards[usr] = 0; emit
    ↪ Deny(usr); }
```

3.8 CVF-8

- **Severity** Minor
- **Category** Documentation
- **Status** Info
- **Source** DssVest.sol

Description The semantics of the keys in this mapping is unclear.
Recommendation Consider adding a documentation comment.
Client Comment We think it is intuitive enough as it is.

Listing 8:

```
mapping (uint256 => Award) public awards;
```

3.9 CVF-9

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** DssVest.sol

Recommendation These two variables could be merged into a single dynamic array of Award structures.
Client Comment We prefer to use the mapping for starting with id=1.

Listing 9:

```
mapping (uint256 => Award) public awards;
uint256 public ids;
```

3.10 CVF-10

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DssVest.sol

Recommendation This variable could be replaced by a flag inside "Award" structure.

Listing 10:

```
mapping (uint256 => uint256) public restricted;
```

3.11 CVF-11

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** DssVest.sol

Recommendation More conventional form would be: return x > y ? y : x;

Listing 11:

```
if (x > y) { z = y; } else { z = x; }
```

3.12 CVF-12

- **Severity** Minor
- **Category** Bad naming
- **Status** Fixed
- **Source** DssVest.sol

Description The function name is confusing. It looks like an initialized of the contract, i.e. a function that is run only once.
Recommendation Consider renaming to something like "award".

Listing 12:

```
function init(address _usr, uint256 _tot, uint256 _bgn, uint256 _tau, uint256 _clf, address _mgr) external auth lock returns (uint256 id) {
```

3.13 CVF-13

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** DssVest.sol

Recommendation This check is redundant. It is anyway possible to specify a dead address as a recipient.
Client Comment We use the check award.usr != address(0) to see if an award exists or not.

Listing 13:

```
require(_usr != address(0),                    "DssVest/invalid-user");

require(_dst != address(0), "DssVest/zero-address-invalid");
```

3.14 CVF-14

- **Severity** Minor
- **Category** Flaw
- **Status** Info
- **Source** DssVest.sol

Recommendation Should probably be "<=".
Client Comment The require had been previously removed due checking almost the same than toUint128()

Listing 14:

```
require(_tot < uint128(-1),                    "DssVest/amount-error");
```

3.15 CVF-15

- **Severity** Minor
- **Category** Readability
- **Status** Info
- **Source** DssVest.sol

Recommendation The maximum uint128 value could be written as "type(uint128).max".
Client Comment Same prev one.

Listing 15:

```
require(_tot < uint128(-1),                          "DssVest/
    ↪ amount-error");
```

3.16 CVF-16

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** DssVest.sol

Description This relies on the fact that it is at least 20 yeas between the epoch and a block timestamp.
Recommendation Consider rewriting like this: add(_bgn, TWENTY_YEARS) > block.timestamp
Client Comment We think is more readable the way it is.

Listing 16:

```
require(_bgn > sub(block.timestamp, TWENTY_YEARS), "DssVest/bgn-
    ↪ too-long-ago");
```

3.17 CVF-17

- **Severity** Minor
- **Category** Flaw
- **Status** Info
- **Source** DssVest.sol

Description This check treats rounding errors towards user.
Recommendation Consider rewriting like this to avoid rounding errors: _tot <= mul(cap, _tau),
Client Comment Precision here is not necessary and we think is easier to reason the way it. It is also defined the same way cap will be. Changing to mul will break the tests.

Listing 17:

```
require(_tot / _tau <= cap,                          "DssVest/rate
    ↪ -too-high");
```

3.18 CVF-18

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** DssVest.sol

Recommendation This check should be done earlier, before the more expensive checks.
Client Comment Gas is not critical here and this will hardly fail anyway as will be done via governance spells. We prefer to keep the order of params in the set of requires.

Listing 18:

```
require(_tau <= TWENTY_YEARS,                          "DssVest/tau-
  ↪ too-long");
```

3.19 CVF-19

- **Severity** Major
- **Category** Unclear behavior
- **Status** Fixed
- **Source** DssVest.sol

Description This check never fails. Should it be "ids < unit256(-1)" ?

Listing 19:

```
require(id < uint256(-1),                              "DssVest/id-
  ↪ overflow");
```

3.20 CVF-20

- **Severity** Minor
- **Category** Readability
- **Status** Fixed
- **Source** DssVest.sol

Recommendation The maximum uint256 value could be written as "type(uint256).max".

Listing 20:

```
144 require(id < uint256(-1),                          "DssVest/id-
  ↪ overflow");

164 _vest(_id, uint256(-1));
```

3.21 CVF-21

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** DssVest.sol

Description This skips the zero ID. Is it intentional? If so, what is the reasoning behind such behavior?
Client Comment It's simpler for an end user to reason from id = 1.

Listing 21:

```
id = ++ids;
```

3.22 CVF-22

- **Severity** Minor
- **Category** Documentation
- **Status** Fixed
- **Source** DssVest.sol

Description For non restircted awards, anybody may actually call this function.
Recommendation Consider fixing the documentation comment.

Listing 22:

```
@dev Owner of a vesting contract calls this to claim all
    ↪ available rewards
```

3.23 CVF-23

- **Severity** Minor
- **Category** Flaw
- **Status** Info
- **Source** DssVest.sol

Description There are no range checks for the "_id" argument.
Recommendation Consider adding proper checks.
Client Comment We use the usr to check award existance.

Listing 23:

```
function _vest(uint256 _id, uint256 _maxAmt) internal {

function accrued(uint256 _id) external view returns (uint256 amt) {

function restrict(uint256 _id) external {

function unrestrict(uint256 _id) external {

function _yank(uint256 _id, uint256 _end) internal {

function move(uint256 _id, address _dst) external {

function valid(uint256 _id) external view returns (bool isValid) {
```

3.24 CVF-24

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** DssVest.sol

Recommendation It is a common good practice to record a payment before actually performing it.

Listing 24:

```
pay(_award.usr, amt);
awards[_id].rxd = toUint128(add(awards[_id].rxd, amt));
```

3.25 CVF-25

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** DssVest.sol

Recommendation The current value of "awards[_id].rxd" is already read from the storage and is available as "_award.rxd". No need to read it again here.

Listing 25:

```
awards[_id].rxd = toUint128(add(awards[_id].rxd, amt));
```

3.26 CVF-26

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** DssVest.sol

Recommendation An invalid award ID could be detected without reading the whole award into the memory, by range checking the ID.
Client Comment If we want to do the range check, we also need to SLOAD ids value, which will penalize good calls in favor to ones that will revert.

Listing 26:

```
197 require(_award.usr != address(0), "DssVest/invalid-award");

225 require(_award.usr != address(0), "DssVest/invalid-award");

285 require(_award.usr != address(0), "DssVest/invalid-award");
```

3.27 CVF-27

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** DssVest.sol

Recommendation This formula is overcomplicated. The conventional way to calculate this proportion would be: _tot * (_time - _bgn) / (_fin - _bgn).

Listing 27:

```
214 uint256 t = mul(sub(_time, _bgn), WAD) / sub(_fin, _bgn); // 0
    ↪ <= t < WAD
amt = mul(_tot, t) / WAD; // 0 <= gem < _award.tot
```

3.28 CVF-28

- **Severity** Minor
- **Category** Flaw
- **Status** Fixed
- **Source** DssVest.sol

Description This formula divides twice, thus it may accumulate rounding errors.
Recommendation Consider dividing only once.

Listing 28:

```
uint256 t = mul(sub(_time, _bgn), WAD) / sub(_fin, _bgn); // 0 <= t < WAD
amt = mul(_tot, t) / WAD; // 0 <= gem < _award.tot
```

3.29 CVF-29

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** DssVest.sol

Recommendation These functions should probably log some events.

Listing 29:

```
function restrict(uint256 _id) external {

function unrestrict(uint256 _id) external {
```

3.30 CVF-30

- **Severity** Major
- **Category** Flaw
- **Status** Fixed
- **Source** DssVest.sol

Recommendation These functions allow modifying not yet created awards.

Listing 30:

```
function restrict(uint256 _id) external {

function unrestrict(uint256 _id) external {
```

3.31 CVF-31

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** DssVest.sol

Description This function actually don't remove the vesting contract, but just modifies it in a certain way.
Recommendation Consider fixing the comment.

Listing 31:

```
@dev Allows governance or the manager to remove a vesting contract
```

3.32 CVF-32

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** DssVest.sol

Recommendation It would be better to just revert in such a case.
Client Comment We prefer not to revert this as yank can be used in a governance spell but the award might change its status in the meanwhile and make the whole spell to be stuck.

Listing 32:

```
} else if (_end > _award.fin) {
    _end = _award.fin;
```

3.33 CVF-33

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** DssVest.sol

Description This may end up in a situation when 'fin < clf'.
Recommendation Consider setting 'clf = _end' in such a case.

Listing 33:

```
awards[_id].fin = toUint48(_end);
```

3.34 CVF-34

- **Severity** Minor
- **Category** Flaw
- **Status** Info
- **Source** DssVest.sol

Recommendation The event is emitted even if the owner didn't actually change.
Client Comment We prefer not adding logic for emitting the event when it's not really needed.

Listing 34:

```
emit Move(_id, _dst);
```

3.35 CVF-35

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** DssVest.sol

Recommendation The type of the argument should be "MintLike".
Client Comment We prefer using standard types as arguments.

Listing 35:

```
constructor(address _gem) public DssVest() {
```

3.36 CVF-36

- **Severity** Minor
- **Category** Readability
- **Status** Info
- **Source** DssVest.sol

Recommendation This value could be rendered as "1e27".
Client Comment We use RAY for explicitness.

Listing 36:

```
uint256 internal constant RAY = 10**27;
```

3.37 CVF-37

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** DssVest.sol

Recommendation The type of the argument should be "ChainlogLike".
Client Comment We prefer using standard types as arguments.

Listing 37:

```
constructor(address _chainlog) public DssVest() {
```

3.38 CVF-38

- **Severity** Minor
- **Category** Readability
- **Status** Fixed
- **Source** DssVest.sol

Description The "_chainlog" argument is casted to "ChainlogLike" three times.
Recommendation Consider refactoring.

Listing 38:

```
chainlog = ChainlogLike(_chainlog);
VatLike _vat = vat = VatLike(ChainlogLike(_chainlog).getAddress
    ↪ ("MCD_VAT"));
DaiJoinLike _daiJoin = daiJoin = DaiJoinLike(ChainlogLike(
    ↪ _chainlog).getAddress("MCD_JOIN_DAI"));
```