ABDK CONSULTING

SMART CONTRACT
FINAL AUDIT

MakerDAO

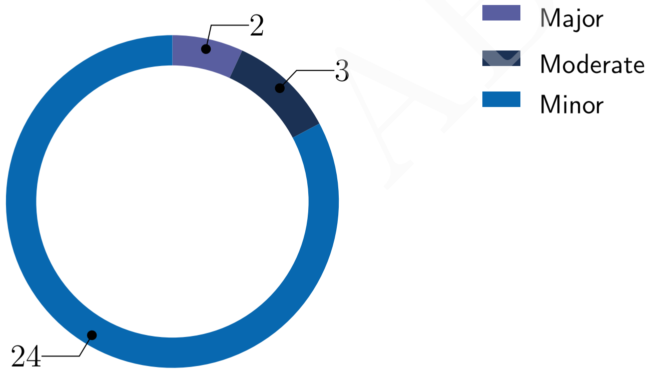
Crop

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
10th April 2021

We've been asked to review a part of MakerDAO smart contracts given in separate files. These contracts deal with dividend distribution from tokens deposited to Maker contracts. We have checked protocol correctness, optimality, scalability, and fairness. We have identified several significant issues arising from rounding errors which may cause the contract malfunction in the long term.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Suboptimal	Opened
CVF-2	Minor	Procedural	Opened
CVF-3	Minor	Procedural	Opened
CVF-4	Minor	Procedural	Opened
CVF-5	Minor	Bad datatype	Opened
CVF-6	Minor	Suboptimal	Opened
CVF-7	Minor	Bad naming	Opened
CVF-8	Minor	Bad datatype	Opened
CVF-9	Minor	Suboptimal	Opened
CVF-10	Minor	Overflow/Underflow	Opened
CVF-11	Moderate	Suboptimal	Opened
CVF-12	Minor	Suboptimal	Opened
CVF-13	Minor	Overflow/Underflow	Opened
CVF-14	Minor	Suboptimal	Opened
CVF-15	Minor	Documentation	Opened
CVF-16	Minor	Suboptimal	Opened
CVF-17	Moderate	Suboptimal	Opened
CVF-18	Major	Flaw	Opened
CVF-19	Major	Flaw	Opened
CVF-20	Moderate	Suboptimal	Opened
CVF-21	Minor	Suboptimal	Opened
CVF-22	Minor	Suboptimal	Opened
CVF-23	Minor	Suboptimal	Opened
CVF-24	Minor	Suboptimal	Opened
CVF-25	Minor	Suboptimal	Opened
CVF-26	Minor	Suboptimal	Opened
CVF-27	Minor	Flaw	Opened

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Opened
CVF-29	Minor	Suboptimal	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	Apr. 10, 2021	D. Khovratovich	Initial Draft
0.2	Apr. 10, 2021	D. Khovratovich	Minor revision
1.0	Apr. 11, 2021	D. Khovratovich	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 have reviewed the contracts in the crop repository, version 1.0.0:

- crop.sol;
- sushi.sol

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.
- **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** Suboptimal
- **Status** Opened
- **Source** crop.sol

Recommendation Should be ^0.6.0 according to a common best practice.

Listing 1:

```
solidity 0.6.12;
```

3.2 CVF-2

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** crop.sol

Description We didn't review this file.

Listing 2:

```
"dss-interfaces/dss/VatAbstract.sol";
```

3.3 CVF-3

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** crop.sol

Recommendation This interface should be moved to a separate file named "ERC20.sol".

Listing 3:

```
ERC20 {
```

3.4 CVF-4

- **Severity** Minor
- **Category** Procedural
- **Status** Opened
- **Source** crop.sol

Recommendation This contract should be moved to a separate file named "CropJoin.sol".

Listing 4:

```
CropJoin {
```

3.5 CVF-5

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** crop.sol

Description In ERC20 decimals property has type "uint8".
Recommendation Consider using the same type here.

Listing 5:

```
uint256      public immutable dec;     // gem decimals
```

3.6 CVF-6

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** crop.sol

Description All decimals handling should probably be delegated to descendant contracts. Currently it just wastes gas for the cases when gem has 18 decimals.

Listing 6:

```
uint256      public immutable dec;     // gem decimals
```

3.7 CVF-7

- **Severity** Minor
- **Category** Bad naming
- **Status** Opened
- **Source** crop.sol

Description The semantics of this event is unclear from its name. The meaning of the last parameter is unclear.
Recommendation Consider renaming the event and/or the parameter, also consider adding a documentation comment.

Listing 7:

```
event Tack(address indexed src, address indexed dst, uint256 wad
    ↪ );
```

3.8 CVF-8

- **Severity** Minor
- **Category** Bad datatype
- **Status** Opened
- **Source** crop.sol

Recommendation The type of the "_vat" parameter should be "VatAbstract", the types of the "_get" and "_bonus" parameters should be "ERC20".

Listing 8:

```
constructor(address vat_, bytes32 ilk_, address gem_, address
    ↪ bonus_) public {
```

3.9 CVF-9

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** crop.sol

Recommendation The value could be specified shorter as "1e18".

Listing 9:

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

3.10 CVF-10

- **Severity** Minor
- **Category** Overflow/Underflow
- **Status** Opened
- **Source** crop.sol

Description Phantom overflow is possible here, i.e. situation when the final result would fit into the destination type, but some intermediary calculations overflow.

Listing 10:

```
z = mul(x, y) / WAD;

z = mul(x, WAD) / y;

z = mul(x, y) / RAY;

z = mul(x, RAY) / y;
```

3.11 CVF-11

- **Severity** Moderate
- **Category** Suboptimal
- **Status** Opened
- **Source** crop.sol

Recommendation The expression "10 ** (18 - dec)" depends only on an immutable variable and should be calculated once in the constructor.

Listing 11:

```
74  return mul(_nav, 10 ** (18 - dec));

88  uint256 wad = wdiv(mul(val, 10 ** (18 - dec)), nps());

109 uint256 wad = wdiv(mul(val, 10 ** (18 - dec)), nps());
```

3.12 CVF-12

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** crop.sol

Recommendation NAV per share is undefined when there is no shares, so this function should probably just revert in such case.

Listing 12:

```
79  if (total == 0) return WAD;
```

3.13 CVF-13

- **Severity** Minor
- **Category** Overflow/Underflow
- **Status** Opened
- **Source** crop.sol

Description Phantom overflow is possible here, i.e. a situation when the final result would fit into the destination type, but some intermediary calculations overflow.

Listing 13:

```
80  else return wdiv(nav(), total);
```

3.14 CVF-14

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** crop.sol

Description This is equivalent to: val * 10**(18 - dec) * 1e18 / (gem.balanceOf (this) * 10**(18 - dec) * 1e18 / total) and could be reduced to: val * total / gem.balanceOf (this).

Listing 14:

```
uint256 wad = wdiv(mul(val, 10 ** (18 - dec)), nps());

uint256 wad = wdiv(mul(val, 10 ** (18 - dec)), nps());
```

3.15 CVF-15

- **Severity** Minor
- **Category** Documentation
- **Status** Opened
- **Source** crop.sol

Description This overflow check looks like a range check.
Recommendation Consider adding a comment explaining what this check actually does, or just remove it and use safe cast to int256 in the code.below.

Listing 15:

```
require(int256(wad) >= 0);

require(int256(wad) >= 0);
```

3.16 CVF-16

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** crop.sol

Description This reward distribution code is duplicated.
Recommendation Consider moving it into a separate function or even into a modifier.

Listing 16:

```
if (total > 0) share = add(share, rdiv(crop(), total));

address usr = msg.sender;
require(bonus.transfer(usr, sub(rmul(stake[usr], share), crops[usr])));
stock = bonus.balanceOf(address(this));

crops[usr] = rmul(stake[usr], share);

if (total > 0) share = add(share, rdiv(crop(), total));

address usr = msg.sender;
require(bonus.transfer(usr, sub(rmul(stake[usr], share), crops[usr])));
stock = bonus.balanceOf(address(this));

crops[usr] = rmul(stake[usr], share);
```

3.17 CVF-17

- **Severity** Moderate
- **Category** Suboptimal
- **Status** Opened
- **Source** crop.sol

Description The ‘stock‘ storage variable is needed only for "push" rewards. Maintaining it for all types of rewards is suboptimal.
Recommendation Consider moving all usages of this storage variable into functions that could be overridden by particular by descendant contracts to avoid excess gas usage.

Listing 17:

```
stock = bonus.balanceOf(address(this));

stock = bonus.balanceOf(address(this));
```

3.18 CVF-18

- **Severity** Major
- **Category** Flaw
- **Status** Opened
- **Source** crop.sol

Description These values are rounded down, i.e. toward the benefit of a user. This could lead to a situation when the contract will not have enough bonus tokens to pay rewards in full to all the users.
Recommendation Consider always rounding towards the benefit of the contract.

Listing 18:

```
103 crops[usr] = rmul(stake[usr], share);

124 crops[usr] = rmul(stake[usr], share);

140 crops[usr] = rmul(stake[usr], share);

150 crops[dst] = add(crops[dst], rmul(share, wad));
```

3.19 CVF-19

- **Severity** Major
- **Category** Flaw
- **Status** Opened
- **Source** crop.sol

Description This value is rounded down, i.e. toward the benefit of a user. This could lead to a situation when the contract will not be able to perform exit operation because total would go negative.
Recommendation Consider always rounding towards the benefit of the contract.

Listing 19:

```
109 uint256 wad = wdiv(mul(val, 10 ** (18 - dec)), nps());
```

3.20 CVF-20

- **Severity** Moderate
- **Category** Suboptimal
- **Status** Opened
- **Source** crop.sol

Recommendation The expression "10 ** dec" depends only on an immutable variable and should be calculated once in the constructor.

Listing 20:

```
133 uint256 val = wmul(wmul(wad, nps()), 10 ** dec);
```

3.21 CVF-21

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** crop.sol

Description This is equivalent to: wad * (gem.balanceOf(this) * 10**(18 - dec) * 1e18 / total) / 1e18 * 10**dec / 1e18 and can be reduced to: wad * gem.balanceOf(this) / total.

Listing 21:

```
uint256 val = wmul(wmul(wad, nps()), 10 ** dec);
```

3.22 CVF-22

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** crop.sol

Description The expression "rmul (share, wad)" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 22:

```
crops[src] = sub(crops[src], rmul(share, wad));
crops[dst] = add(crops[dst], rmul(share, wad));
```

3.23 CVF-23

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** sushi.sol

Recommendation Should be "Ô.6.0" according to a common best practice.

Listing 23:

```
solidity 0.6.12;
```

3.24 CVF-24

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** sushi.sol

Recommendation This interface should be moved to a separate file named "MasterChefLike.sol".

Listing 24:

```
MasterChefLike {
```

3.25 CVF-25

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** sushi.sol

Recommendation This contract should be moved to a separate file named "SushiJoin.sol".

Listing 25:

```
SushiJoin is CropJoin {
```

3.26 CVF-26

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** sushi.sol

Recommendation This event is emitted even if the status is not changed.

Listing 26:

```
emit Rely(usr);

emit Deny(usr);
```

3.27 CVF-27

- **Severity** Minor
- **Category** Flaw
- **Status** Opened
- **Source** sushi.sol

Recommendation The "_dem" argument should have type "ERC20", and the "masterchef_" argument should have type "MasterChefLike".

Listing 27:

```
constructor(address vat_, bytes32 ilk_, address gem_, address
    ↪ bonus_, address masterchef_, uint256 pid_)
```

3.28 CVF-28

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** sushi.sol

Description This function is not called in this contract. Probably not an issue.

Listing 28:

```
function crop() internal override returns (uint256) {
```

3.29 CVF-29

- **Severity** Minor
- **Category** Suboptimal
- **Status** Opened
- **Source** sushi.sol

Description This expression will be evaluated again inside "super.flee()" call.
Recommendation Consider refactoring the code to avoid double execution.

Listing 29:

```
uint256 val = vat.gem(ilk, msg.sender);
```