ABDK CONSULTING

SMART CONTRACT AUDIT

Maker DAO

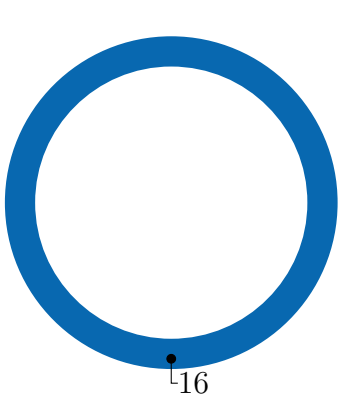
Vote Delegate

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
8th June 2021

We've been asked to review the Vote Delegate smart contracts given in separate files.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Opened
CVF-2	Minor	Procedural	Opened
CVF-3	Minor	Procedural	Opened
CVF-4	Minor	Bad naming	Opened
CVF-5	Minor	Bad naming	Opened
CVF-6	Minor	Suboptimal	Opened
CVF-7	Minor	Bad datatype	Opened
CVF-8	Minor	Readability	Opened
CVF-9	Minor	Flaw	Opened
CVF-10	Minor	Readability	Opened
CVF-11	Minor	Procedural	Opened
CVF-12	Minor	Bad naming	Opened
CVF-13	Minor	Bad datatype	Opened
CVF-14	Minor	Procedural	Opened
CVF-15	Minor	Suboptimal	Opened
CVF-16	Minor	Suboptimal	Opened

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	June 6, 2021	D. Khovratovich	Initial Draft
0.2	June 6, 2021	D. Khovratovich	Minor revision
1.0	June 7, 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.

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 About Customer

is a

2.3 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.

2.4 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
- **Status** Opened
- **Category** Procedural
- **Source** VoteDelegate.sol

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

Listing 1:

```
19 solidity 0.6.12;
```

3.2 CVF-2

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

Recommendation These interfaces should be moved to separate files named "TokenLike.sol" and "ChiefLike.sol" respectively.

Listing 2:

```
21 TokenLike {

30 ChiefLike {
```

3.3 CVF-3

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

Description These functions are not used.
Recommendation Consider removing them.

Listing 3:

```
22 function balanceOf(address) external view returns (uint256);

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

3.4 CVF-4

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

Description These functions are not common for tokens in general, so the interface most probably represents not a generic token, but some specific flavor of tokens, which should be reflected in the interface name.

Listing 4:

```
24 function pull(address, uint256) external;
   function push(address, uint256) external;

27 function mint(address, uint256) external;

33 function approvals(address) external view returns (uint256);
   function deposits(address) external view returns (uint256);
```

3.5 CVF-5

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

Description The meaning of the keys and values in this mapping is unclear.
Recommendation Consider adding a documentation comment.

Listing 5:

```
42 mapping(address => uint256) public delegators;
```

3.6 CVF-6

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

Recommendation These variables don't have to be public as their values could be derived from the value of the "chief" variable.

Listing 6:

```
44 TokenLike public immutable gov;
   TokenLike public immutable iou;
```

3.7 CVF-7

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

Recommendation The "_chief" argument should have type "ChiefLike".

Listing 7:

```
48 constructor(address _chief, address _delegate) public {
```

3.8 CVF-8

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** VoteDelegate.sol

Recommendation "type(uint256).max" would be more readable than "uint256(-1)".

Listing 8:

```
55 ChiefLike(_chief).GOV().approve(_chief, uint256(-1));
   ChiefLike(_chief).IOU().approve(_chief, uint256(-1));
```

3.9 CVF-9

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

Recommendation These function should probably log some events.

Listing 9:

```
71 function lock(uint256 wad) external {

78 function free(uint256 wad) external {
```

3.10 CVF-10

- **Severity** Minor
- **Status** Opened
- **Category** Readability
- **Source** VoteDelegate.sol

Description Here a safe subtraction is used to enforce a business-level constraint. This is error-prone, makes the code harder to read, and produces misleading error messages on underflow.
Recommendation Consider checking buslness-level constraints explicitly.

Listing 10:

```
79 delegators[msg.sender] = sub(delegators[msg.sender], wad);
```

3.11 CVF-11

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

Recommendation Should be "ĝ0.6.0" according to a common best practice.

Listing 11:

```
18 solidity 0.6.12;
```

3.12 CVF-12

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

Recommendation Events are usually named via nouns, such as "NewVoteDelegate" or just "VoteDelegate", and "VoteDelegateDestruction".

Listing 12:

```
26 event VoteDelegateCreated(

31 event VoteDelegateDestroyed(
```

3.13 CVF-13

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

Recommendation The argument type should be "ChiefLike".

Listing 13:

```
36 constructor(address chief_) public {
```

3.14 CVF-14

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

Recommendation The surrounding brackets are redundant.

Listing 14:

```
41 return (address(delegates[guy]) != address(0));
```

3.15 CVF-15

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

Description This check doesn't make much sense, as it could be easily overridden by calling the "destroy" function before the "create" function.
Recommendation Consider removing this check or removing the "destroy" function.

Listing 15:

```
45 require(!isDelegate(msg.sender), "this address is already a
   ↪ delegate");
```

3.16 CVF-16

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

Recommendation This function doesn't make much sense, as it doesn't actually destroy the 'VoteDelegate' contract, but just removes the link to it from the 'VoteDelegateFactory' contract.

Listing 16:

```
52 function destroy() external {
```