HACKEN

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer**: SingularDTV
**Date**: June 17, 2020
**Platform**: Ethereum
**Language**: Solidity

This document may contain confidential information about IT systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.
The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities fixed - upon decision of customer.

## Document

| | |
|---|---|
| **Name** | Smart Contract Code Review and Security Analysis Report for SingularDTV |
| **Platform** | Ethereum / Solidity |
| **Repository** | https://github.com/SingularDTV/snglsDAO-smartcontracts/tree/master/dao-contracts |
| **Commit** | d453ff098d6562b469a4edda898708ee6dde9dd2 |
| **Branch** | master |
| **Date** | 17.06.2020 |

# Table of contents

## Introduction

Hacken OÜ (Consultant) was contracted by SingularDTV (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contract and its code review conducted between June 3rd, 2020 - June 17th, 2020.

## Scope

The scope of the project are smart contracts, selected by Customer in dao-contracts repository:
1. DAOstack contracts - contracts that were developed by DAOstack. They were already audited and are considered secure. However, these DAOstack contracts are in scope of this audit:

- Avatar.sol
- DAOToken.sol
- Reputation.sol
- GenericScheme.sol

2. DAOstack Schemes contracts that were changed by SingularDTV:

- LockingSGT4Reputation.sol
- ContributionReward.sol

3. Standalone and self-developed contracts:

- Fee.sol
- MembershipFeeStaking.sol

Repository - https://github.com/SingularDTV/snglsDAO-smartcontracts/tree/master/dao-contracts
Commit - d453ff098d6562b469a4edda898708ee6dde9dd2
Branch - master

**Note.** All other contracts that are present in the repository were developed by DAOStack and already audited. They are considered secure in this audit and they are out-of-scope of the audit.

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

## Executive Summary

According to the assessment, Customer`s smart contracts are secured.

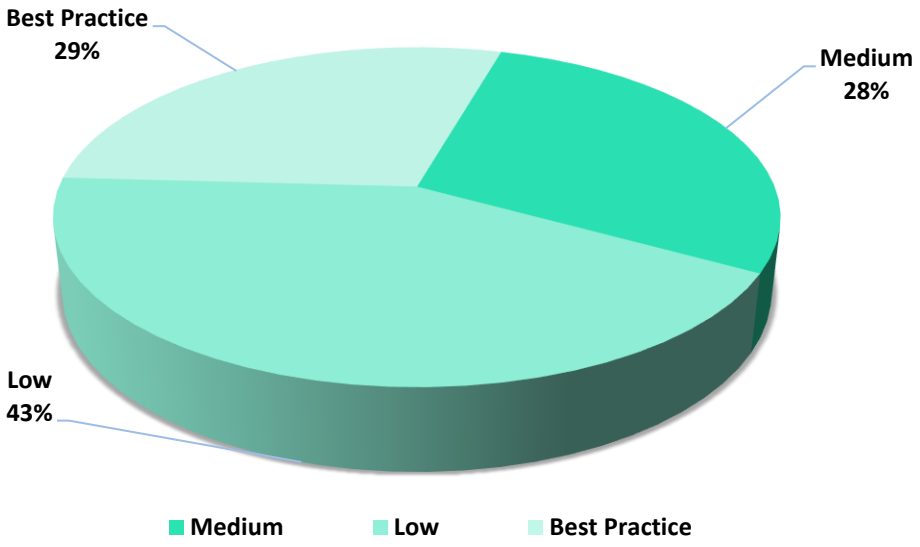
| Insecure | Poor secured | Secured | Well-secured |
|---|---|---|---|

You are here

Our team performed analysis of code functionality, manual audit and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in Audit overview section. General overview is presented in AS-IS section and all found issues can be found in Audit overview section.

We found 2 medium, 3 low and 2 best practice issues in smart contract code.

Graph 1. The distribution of vulnerabilities.



## Severity Definitions

| Risk Level | Description |
|---|---|
| **Critical** | Critical vulnerabilities are usually straightforward to exploit and can lead to tokens lose etc. |
| **High** | High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions |
| **Medium** | Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose |
| **Low** | Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution |
| **Lowest / Code Style / Best Practice** | Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored. |

## AS-IS overview

**Avatar.sol**

**Avatar** is the contract that stores all assets of the DAO.

**Avatar** contract is **Ownable**.

**Ownable** is access control contract that defines owner role and considered secure.

Contract **Avatar** defines following public parameters:

- string **orgName**
- DAOToken **nativeToken**
- Reputation **nativeReputation**

**Note.** **Reputation** and **DAOToken** contracts are reviewed below

Contract **Avatar** has 8 functions:

- **constructor** - sets **orgName**, **nativeToken** and **nativeReputation**
- **fallback** - external payable function to receive ether
- **genericCall** - public function that performs generic call to specified contract. Has **onlyOwner** modifier
- **sendEther** - public function that sends ether to specified receiver. Has **onlyOwner** modifier
- **externalTokenTransfer** - public function that transfers ERC20 token to specified receiver. Has **onlyOwner** modifier
- **externalTokenTransferFrom** - public function that performs transferFrom for ERC20 token to receiver. Has **onlyOwner** modifier
- **externalTokenApproval** - public function that approves transfer of ERC20 token for spender. Has **onlyOwner** modifier
- **metadata** - public function that emits Metadata event. Has **onlyOwner** modifier

**DAOToken.sol**

**DAOToken** is the contract that describes DAO ERC20 token

**DAOToken** contract is **ERC20**, **ERC20Burnable**, **Ownable**.

**ERC20**, **ERC20Burnable**, **Ownable** are known contracts that are well audited and considered secure.

Contract **DAOToken** defines following public parameters:

- string **name**
- string **symbol**
- uint8 constant **decimals** that is set to 18
- uint256 **cap**

Contract **DAOToken** has 2 functions:

- **constructor** - sets **name**, **symbol** and **cap**
- **mint** - public function that mints tokens to account. Has **onlyOwner** modifier

**Reputation.sol**

**Reputation** is the contract that manages the reputation for DAO users.

**Reputation** contract is **Ownable**.

**Ownable** is access control contract that defines owner role and considered secure.

Contract **Reputation** defines following parameters and structures:

- uint8 public **decimals** that is set to 18
- struct **Checkpoint** that stores uint128 **fromBlock**, uint128 **value**
- private mapping (**address** => **Checkpoint[]**) **balances**
- **Checkpoint[]** private **totalSupplyHistory**

Contract **Reputation** has 8 functions:

- **mint** - public function that mints tokens to account. Has **onlyOwner** modifier
- **burn** - public function that burns tokens from account. Has **onlyOwner** modifier
- **totalSupply** - public view function that returns total amount of reputation on the current block
- **balanceOf** - public view function that returns account's amount of reputation on the current block
- **totalSupplyAt** - public view function that returns total amount of reputation on the specified block
- **balanceOfAt** - public view function that returns account's amount of reputation on the specified block
- **getValueAt** - internal view function that returns the amount of reputation in specified block for the checkpoints array
- **updateValueAtNow** - internal function that updates reputation amount for checkpoint array

**GenericScheme.sol**

**GenericScheme** is the contract that responsible for proposing and executing calls for organization avatar.

**GenericScheme** contract imports **IntVoteInterface.sol**, **VotingMachineCallbacksInterface.sol**, **VotingMachineCallbacks.sol** that are considered secure.

Contract **GenericScheme** is **VotingMachineCallbacks**, **ProposalExecuteInterface**.

Contract **GenericScheme** defines following parameters and structures:

- struct **CallProposal** that stores bytes **callData**, uint256 **value**, bool **exist**, bool **passed**
- public mapping (**bytes32**=>**CallProposal**) **organizationProposals**
- public IntVoteInterface **votingMachine**
- public bytes32 **voteParams**
- public address **contractToCall**
- public Avatar **avatar**;

Contract **GenericScheme** has 4 functions:

- **initialize** - external function that initializes contract parameters with specified values
- **executeProposal** - external function that marks proposal as passed or deletes it by voting machine contract. Has **onlyVotingMachine(_proposalId)** modifier
- **execute** - public function that executes the proposal approved by voting machine
- **proposeCall** - public function that creates new proposal

### LockingSGT4Reputation.sol

**LockingSGT4Reputation** is the contract that responsible for minting and burning reputation.

**MembershipFeeStaking** contract imports **IERC20.sol**, **SafeMath.sol**, **Controller.sol** that are considered secure.

Contract **MembershipFeeStaking** defines following parameters and structures:

- struct **Locker** that stores uint256 **amount**, uint256 **releaseTime**
- public Avatar **avatar**
- public IERC20 **sgtToken**
- public mapping (**address** => **Locker**) **lockers**
- public uint256 **totalLocked**
- public uint256 **minLockingPeriod**

Contract **MembershipFeeStaking** has 3 functions:

- **release** - public function that releases locked tokens and burns reputation
- **lock** - public function that locks tokens and mints reputation
- **initialize** - public function that initializes contract parameters with specified values

### ContributionReward.sol

**ContributionReward** is the contract that responsible for proposing and rewarding contributions to organizations.

**ContributionReward** contract imports **IntVoteInterface.sol**, **VotingMachineCallbacksInterface.sol**, **UniversalScheme.sol** and **VotingMachineCallbacks.sol** that are considered secure.

contract **ContributionReward** is **UniversalScheme**, **VotingMachineCallbacks**, **ProposalExecuteInterface**.

Contract **ContributionReward** defines following parameters and structures:

- struct **ContributionProposal** that stores uint256 **nativeTokenReward**, int256 **reputationChange**, uint256 **ethReward**, IERC20 **externalToken**, uint256 **externalTokenReward**, address payable **beneficiary**, uint256 **periodLength**, uint256 **numberOfPeriods**, uint256 **executionTime**, uint256[4] **redeemedPeriods**
- public mapping (**address** => mapping (**bytes32** => **ContributionProposal**)) **organizationsProposals**
- struct **Parameters** that stores bytes32 **voteApproveParams**, IntVoteInterface **intVote**
- public mapping (**bytes32** => **Parameters**) **parameters**

Contract **ContributionReward** has 16 functions:

- **executeProposal** - external function that executes proposal by voting machine. Has **onlyVotingMachine(_proposalId)** modifier

- **setParameters** - public function that saves new parameters
- **proposeContributionReward** - public function that submits new proposal for rewarding the contributor
- **redeemReputation** - public function that allows contributor to redeem his reputation
- **redeemNativeToken** - public function that allows contributor to redeem DAO token
- **redeemEther** - public function that allows contributor to redeem Ether
- **redeemExternalToken** - public function that allows contributor to redeem specified ERC20 token
- **redeem** - public function that allows contributor to redeem all rewards
- **getPeriodsToPay** - public view function that returns number of periods left to pay the specified reward
- **getRedeemedPeriods** - public view function that returns number of periods that were already payed of the specified reward
- **getProposalEthReward** - public view function that returns the Ethereum reward for contribution
- **getProposalExternalTokenReward** - public view function that returns the token reward for contribution
- **getProposalExternalToken** - public view function that returns the address of external token that is reward for contribution

- **getProposalExecutionTime** - public view function that returns execution time of the proposal
- **getParametersHash** - public pure function that returns the hash of specified parameters
- **validateProposalParams** - private pure function that validates the params of proposal

**Fee.sol**

**Fee** is the contract that responsible for storing DAO fees.

**Fee** contract is **Ownable**.

**Ownable** is access control contract that defines owner role and considered secure.

Contract **Fee** defines following parameters and structures:

- public uint256 **listingFee**
- public uint256 **transactionFee**
- public uint256 **validationFee**
- public uint256 **membershipFee**

Contract **Fee** has 5 functions:

- **constructor** - sets **listingFee**, **transactionFee**, **validationFee** and **membershipFee**
- **setListingFee** - public function that sets new **listingFee**. Has **onlyOwner** modifier

- **setTransactionFee** - public function that sets new **transactionFee**. Has **onlyOwner** modifier
- **setValidationFee** - public function that sets new **validationFee**. Has **onlyOwner** modifier
- **setMembershipFee** - public function that sets new **membershipFee**. Has **onlyOwner** modifier

**MembershipFeeStaking.sol**

**MembershipFeeStaking** is the contract that responsible for locking and unlocking ERC20 token.

**MembershipFeeStaking** contract imports **IERC20.sol**, **SafeMath.sol** that are considered secure.

Contract **MembershipFeeStaking** defines following parameters and structures:

- struct **Locker** that stores uint256 **amount**, uint256 **releaseTime**
- public Avatar **avatar**
- public IERC20 **sgtToken**
- public mapping (**address** => **Locker**) **lockers**
- public uint256 **totalLocked**
- public uint256 **minLockingPeriod**

Contract **MembershipFeeStaking** has 3 functions:

- **release** - public function that releases locked tokens

- **lock** - public function that locks tokens
- **initialize** - public function that initializes contract parameters with specified values

# Audit overview

## Critical

No critical issues were found.

## High

No high issues were found.

## Medium

1. No gas optimizations for ContributionReward contract. DAOStack's contract was changed by adding 2 checks for proposal validation - proposal can't change reputation; proposal can't reward contributor with SGT token. However, other functionality for reputation change and SGT token reward is present in the contract.

   For example, redeemReputation and redeemNativeToken will always fail and just burn gas if called; redeem function may call redeemReputation and redeemNativeToken, which will just burn gas.

   Moreover, each ContributionProposal stores reputationChange, nativeTokenReward, redeemedPeriods for reputation and SGT token, however, this functionality is disabled so caller would spend extra gas to store constant values. Altogether they need memory for 3 uint256 and 1 int256.

2. totalLocked amount is never decreased in MembershipFeeStaking.sol and LockingSGT4Reputation.sol. When release function is called, totalLocked should be decreased by number of tokens sent to caller.

## Low

3. Solidity version is not locked to the latest stable version. There are also contracts that have different compiler versions in pragma. It is recommended to lock the pragma.

4. The main contract in LockingSGT4Reputation.sol is called MembershipFeeStaking (same name as contract in MembershipFeeStaking.sol). It's recommended to change the contract name to be same as *.sol file name.

5. Comment documentation of release (MembershipFeeStaking.sol line 31; LockingSGT4Reputation.sol line 33) and lock (MembershipFeeStaking.sol line 61; LockingSGT4Reputation.sol line 71) functions says that these functions should return bool, however, they don't return anything. It's recommended to align functionality with documentation.

## Lowest / Code style / Best Practice

6. proposeContributionReward function comment description misses @param _externalToken (line 146), however, it is present in DAOStack version of the code.

7. Fee.sol contract is not documented in the code. It's recommended to add comments that document smart contract functionality.

## Conclusion

Smart contracts within the scope was manually reviewed and analyzed with static analysis tools. For the contract high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Overall quality of reviewed contracts is good. Security engineers found several medium to low issues that don’t have serious security impact.

# Disclaimers

## Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

## Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.