SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: DAO maker
Date: March 8th, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Dao Maker
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Rewards pool
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/daomaker/staking-contract-new/
Commit	2144f6b0af21786be5ff96d42f2737d79cab3275
Timeline	04 MAR 2021 - 08 MAR 2021
Changelog	05 MAR 2021 - INITIAL AUDIT 08 MAR 2021 - SECOND REVIEW.

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by DAO Maker (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 March 3rd, 2021 - March 5th, 2021.

Second review conducted on Match 8th, 2021.

Scope

The scope of the project is smart contracts in the repository:

```
Contract deployment address:
Repository: https://github.com/daomaker/staking-contract-new/
Commit: 2144f6b0af21786be5ff96d42f2737d79cab3275
Files: Farm.sol, FarmManager.sol
```

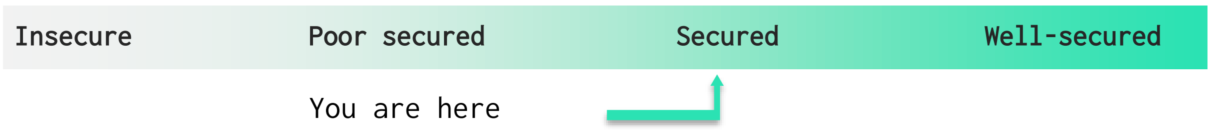
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks

	▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Asset’s integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are secure. Though one issue that can be exploited in a case of the ownership takeover exist.



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

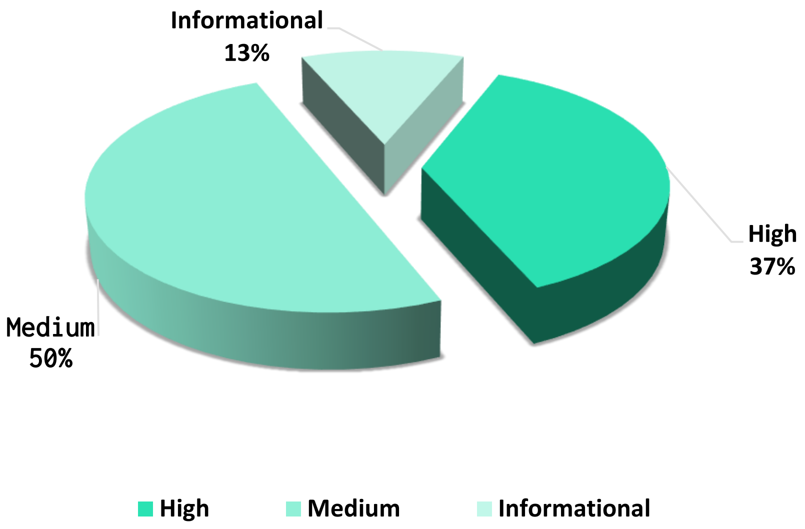
Security engineers found **3** high, **4** medium, and **1** informational issue during the audit.

After the second review Customers` smart contracts contains **1** high severity issues.

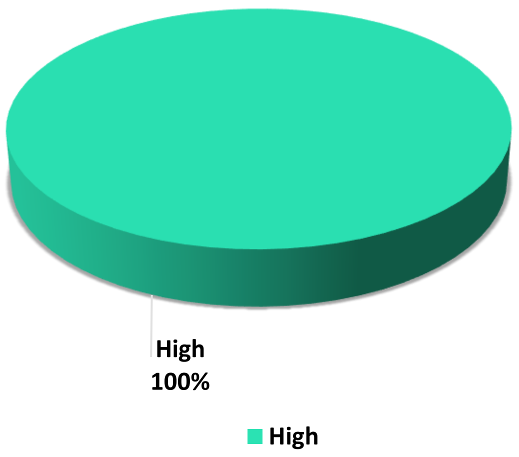
Notice:

1. The Farm contract may be stopped by owners.

Graph 1. Distribution of vulnerabilities after the initial audit.



Graph 2. Distribution of vulnerabilities after the second review.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a 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 assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Informational / Code Style / Best Practice	Informational vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

FarmManager.sol

Description

FarmManager manages *Farm* contracts.

Inheritance

FarmManager contract is Ownable.

Usages

FarmManager contract has following usages:

- SafeERC20 for IERC20
- SafeMath for uint25

Structs

FarmManager contract has no custom structures.

Enums

FarmManager contract has no custom enums.

Events

FarmManager contract has one custom event:

- FarmAdded

Modifiers

FarmManager has no custom modifier.

Fields and constants

FarmManager contract has following fields:

- IFarm[] public farms;
- IERC20[] public stakingTokens
- mapping(address => bool) public funders
- uint public moveBurnRate = 5
- uint public burnRate = 100
- uint public unstakeEpochs = 10
- bool public paused
- address public redistributor

Functions

FarmManager has following external functions:

- *constructor*
  Description
  Initializes the contract. Sets a deployer as funder and redistributor.
  Visibility
  None
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
  None
- *newFarm*
  Description
  Add a new *farm* to the manager. Adds all existing staking tokens to farm.
  Visibility
  public
  Input parameters
    - IFarm farm
  Constraints
    - Can only be called by the owner.
  Events emit
  None
  Output
  None
- *add*
  Description
  Add a new staking token to the manager. Also adds to all existing farms.
  Visibility
  public
  Input parameters
    - uint allocPoint
    - IERC20 stakingToken
  Constraints
    - Can only be called by the owner.
  Events emit
  None
  Output
  None
- *set*
  Description

Update allocation point of a pool.

Visibility

public

Input parameters

- uint allocPoint
- uint _fid
- uint _pid
- bool _withUpdate

Constraints

- Can only be called by the owner.

Events emit

None

Output

None

- *fund*

Description

Fund a farm with *amount*. must give allowance to created farm first.

Visibility

public

Input parameters

- uint _fid
- uint256 _amount

Constraints

- Can only be called by the owner.
- An allowance should be set for a farm contract.

Events emit

None

Output

None

- *changePool*

Description

Allow stakers within a pool to move their stakes.

Visibility

public

Input parameters

- uint _currentFid
- uint _nextFid
- uint _pid

Constraints

- Stake amount should be greater than 0.
- Unstake amount should be 0.
- Withdrawal should not be requested.

Events emit

None

Output

None

- *emergencyWithdrawRewards*
  Description
  Withdraws all reward tokens.
  Visibility
  public
  Input parameters
  None
  Constraints
    - Can only be called by the owner.

  Events emit
  None
  Output
  None
- *updateFunders, setMoveBurnRate, setBurnRate, setUnstakeEpochs, setPaused, setRedistributor*
  Description
  Simple setter function with only owner access.
- *getRedistributor, getMoveBurnRate, getBurnRate, getUnstakeEpochs, getPaused*
  Description
  Simple getters.

Farm.sol

Description

Farm is a liquidity pool with rewards in ERC-20 tokens.

Inheritance

Farm does not inherit anything.

Usages

Farm contract has following usages:

- SafeMath for uint256
- SafeERC20 for IERC20

Structs

Farm contract has following data structures:

- UserInfo
- PoolInfo

Enums

Farm contract has no enums.

Events

Farm contract has following events:

- Deposit
- Withdraw
- Claim
- Unstake
- Initialize

Modifiers

Farm has no custom modifiers.

Fields

Farm contract has following fields and constants:

- IERC20 public erc20
- uint256 public paidOut = 0
- uint256 public rewardPerBlock
- IFarmManager public manager
- PoolInfo[] public poolInfo
- mapping (uint256 => mapping (address => UserInfo)) public userInfo
- uint256 public totalAllocPoint = 0
- uint256 public startBlock
- uint256 public endBlock
- uint256 public constant SECS_EPOCH = 86400

Functions

Farm has following public functions:

- *constructor*
  Description
  Sets initial values of the contract.
  Visibility
  public
  Input parameters
    - IERC20 _erc20
    - uint256 _rewardPerBlock
    - uint256 _startBlock
    - address _manager
  Constraints
  None
  Events emit
  Emits the Initialize event.

Output
None

- *add*
  Description
  Add a new lp to the pool.
  Visibility
  public
  Input parameters
    - uint256 _allocPoint
    - IERC20 _stakingToken
    - bool _withUpdate
  Constraints
    - Can only be called by the FarmManager.
  Events emit
  None
  Output
  None
- *set*
  Description
  Update the given pool's allocation point
  Visibility
  public
  Input parameters
    - uint256 _pid
    - uint256 _allocPoint
    - bool _withUpdate
  Constraints
    - Can only be called by the FarmManager.
  Events emit
  None
  Output
  None
- *massUpdatePools*
  Description
  Update reward variables for all pools.
  Visibility
  public
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
  None
- *updatePool*

Description
Update reward variables of the given pool to be up-to-date.
Visibility
public
Input parameters
  - uint256 _pid

Constraints
None
Events emit
None
Output
None

- *move*

  Description
  Moves LP tokens to another farm.
  Visibility
  external
  Input parameters
    - uint256 _pid

  Constraints
    - Can only be called from the FarmManager.

  Events emit
  Emits the Withdraw event.
  Output
  None

- *deposit*

  Description
  Deposit LP tokens.
  Visibility
  external
  Input parameters
    - uint256 _pid
    - uint256 _amount

  Constraints
    - The contract should not be paused.
    - Unstake should not be requested.

  Events emit
  Emits the Deposit event.
  Output
  None

- *withdraw*

  Description
  Creates a request to unstake all LP tokens.
  Visibility
  external
  Input parameters
    - uint256 _pid

Constraints
  - The contract should not be paused.
  - A message sender should have active balance.
  - Should not be requested yet.

Events emit
Emits the Withdraw event.
Output
None

- *unstake*
  Description
  Withdraw LP tokens. Fee may be applied if unstakeEpochs did not passed yet.
  Visibility
  external
  Input parameters
    - uint256 _pid

  Constraints
    - Unstake should not be requested.

  Events emit
  Emits the Unstake event.
  Output
  None
- *claim*
  Description
  Claims LP tokens from Farm.
  Visibility
  external
  Input parameters
  None
  Constraints
  None
  Events emit
    - The contract should not be paused yet.

  Output
  None
- *emergencyWithdraw*
  Description
  Allows the FarmManager contract to withdraw all rewards to a tx origin.
  Visibility
  public
  Input parameters
  None
  Constraints
  None
  Events emit
    - Can only be called by the FarmManager.

Output
None

- *poolLength*
  Description
  Returns a number of LPs.
- *deposited*
  Description
  Returns deposited amount of a user to a pool.
- *pending*
  Description
  Returns total rewards that have to be payd to a used for a specified pid.
- *totalPending*
  Description
  Returns total rewards that have to be paid to all users.
- *getUserInfo*
  Description
  Returns a user info.

Audit overview

🟥🟥🟥🟥 Critical

No critical issues were found.

🟨🟨🟨 High

1. Owners can set up any number of unstake epochs and any burn rate. As a result, users may lose all their funds when the *unstake* function is called.
   Fixed before the second review. Upper limits for all mentioned values are introduced.

2. Contracts allows to withdraw all reward tokens from all Farms with a single transaction using the owner address. This issue may be exploited in a case of ownership takeover.

🟩🟩 Medium

1. The *newFarm* function of the *FarmManager* has no validation for a farm existence. As a result, farms may be duplicated, and the system may become inconsistent.
   Fixed before the second review.

2. The *add* function of the *Farm* contract has no validation for a staking contract existence. As a result the contract may become inconsistent.
   Fixed before the second review.

3. The *deposit* function of the *Farm* contract operates with the tx.origin instead of the msg.sender to allow moving from one farm to another. As a result the contract may not be used by other contracts.
   Fixed before the second review.

4. The unstakeAmount field of the UserInfo struct is redundant and may be removed or replaced with boolean value for a case when withdraw is requested.
   Fixed before the second review.

🟦 Low

No low severity issues were found.

Informational/ Code style / Best Practice

1. Some code-style issues were found by the static code analyzer.

Conclusion

Smart contracts within the scope were 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.

Security engineers found **3** high, **4** medium, and **1** informational issue during the audit.

After the second review Customers` smart contracts contains **1** high severity issues.

Violations in the following categories were found and addressed to Customer:

Category	Check Item	Comments
Code review	■ Asset's integrity	■ All reward may be withdrawn by owners in one transaction

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.