#HashEx
BLOCKCHAIN SECURITY

Mula Finance

smart contracts final audit report

May 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Mula Finance team to perform an audit of their smart contract. The audit was conducted between 2022-05-05 and 2022-05-06.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at @mula-finance/mula-token GitHub repository after the c2fcde6 commit.

Recheck was made on the e3b2e77 commit.

2.1 Summary

Project name	Mula Finance
URL	https://mulatoken.finance/
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
MulaToken	
MulaIdo & MulaSeed	
MulaIdoVestor & MulaSeedVestor	
MulaTokenUtils	
MulaSaleUtils	
MulaVestorUtils	
Multiple contracts	

3. Found issues

High	6 (23%)
Medium	5 (19%)
Low	12 (46%)
Info	3 (12%)

C1. MulaToken

ID	Severity	Title	Status
C1-01	Low	Code style & gas optimizations	Resolved

C2. MulaIdo & MulaSeed

ID	Severity	Title	Status
C2-01	Medium	Vesting schedule should be personal	Resolved
C2-02	Medium	Price is user-defined	Resolved
C2-03	Low	Code style & gas optimizations	Partially fixed
C2-04	Low	Redundant code	Resolved
C2-05	Low	Unsafe math	Resolved
C2-06	Low	ReentrancyGuard usage	Resolved

C3. MulaIdoVestor & MulaSeedVestor

ID	Severity	Title	Status
C3-01	High	recordInvestment() must not be called after first unlock	Resolved
C3-02	High	Access to recordInvestment() function	Resolved
C3-03	Medium	Contract balance may be insufficient	Resolved
C3-04	Low	Code style & gas optimizations	Resolved
C3-05	Low	Unsafe math	Resolved

C4. MulaTokenUtils

ID	Severity	Title	Status
C4-01	High	Owner exaggerated rights	Resolved
C4-02	Medium	Whitelisting is one-way	Resolved
C4-03	Info	Typos	Resolved
C4-04	Info	Code style & gas optimizations	Partially fixed

C5. MulaSaleUtils

ID	Severity	Title	Status
C5-01	High	Oracle address manipulation	Resolved
C5-02	Low	Division before multiplication	Resolved

C5-03	Low	Code style & gas optimizations	Resolved
C5-04	Low	Price factor	Acknowledged

C6. MulaVestorUtils

ID	Severity	Title	Status
C6-01	High	Token address must be immutable	Resolved
C6-02	High	Missing authorization	Resolved
C6-03	Low	Unsafe math is used	Resolved
C6-04	Low	Code style & gas optimizations	Resolved

C7. Multiple contracts

ID	Severity	Title	Status
C7-01	Medium	Too wide pragma range	Partially fixed
C7-02	Info	Lack of events	Partially fixed

4. Contracts

C1. MulaToken

Overview

ERC-20 standard token with additional transfer restrictions: whitelist is applied until the owner chooses to release the transfers. Relies on the MulaTokenUtils contract.

Issues

C1-01 Code style & gas optimizations

Low Resolved

The `tSupply` variable should be declared `constant`.

No explicit visibility is specified for the `tSupply` variable.

C2. MulaIdo & MulaSeed

Overview

Sale contract that takes payment in native currency or a single ERC-20 token (USD). Inherits the MulaSaleUtils contract. Purchased tokens are vested automatically with MulaIdoVestor/ MulaSeedVestor contracts.

Issues

C2-01 Vesting schedule should be personal

Medium Resolved

The `setVestingDates()` function of MulaVestorUtils changes the global unlocking schedule. A malicious/hacked owner is able to freeze the funds for eternity.

Recommendation

We recommend fixing the vesting terms personally for every user at the moment of sale.

C2-02 Price is user-defined

Medium Resolved

Both the `participateBNB()` and the `participateUSDT()` functions use the user-provided historical (within time frame) BNB/USDT price defined by the `_roundId` parameter.

Recommendation

This behaviour should be either properly documented or eliminated completely. Any possible wrong returned data from the oracle could be exploited during the sale by an arbitrary user.

C2-03 Code style & gas optimizations

Low Partially fixed

No explicit visibility is specified for variables `_tokenContract`, `_vestor`, and `_receiving[][]`.

`_tokenContract` and `_vestor` variables should be declared immutable.

Constrcuctor parameters could be checked for validity. MulaIdoVestor `_vestor` could be deployed during the construction.

Mappings `_receiving[][]` and `CrowdsaleStageBalance[]` could be reduced to a single mapping `_receiving[]` and simple variable `CrowdsaleStageBalance` as only one stage `CrowdsaleStage.PublicSale` is allowed to fill these mappings.

No error messages in require statements in the `_processParticipationBNB()` function.

C2-04 Redundant code

Low Resolved

The `participateUSDT()` function calculates the output amount with excessive operations (USD -> BNB -> sale tokens), which may result in increased division errors. We recommend avoiding double conversion by using the `MulaSaleUtils._rate()` function.

```
function participateUSDT(uint80 _roundId) public  onlyWhileOpen returns(bool){
    (...)
    //calculate number of tokens
    uint256 bnbEquv = convertUsdToBNB(usdVal, _roundId);
    uint256 _numberOfTokens = _MulaReceiving(bnbEquv,_roundId);
    (...)
}
```

C2-05 Unsafe math

Low Resolved

The `_processParticipationBNB()` function uses simple addition besides, the contract has the SafeMath library imported. We recommend uniforming all mathematical operations to either safe by pragma >0.8.0 or safe by using the imported library.

C2-06 ReentrancyGuard usage

Low Resolved

Using the `nonReentrant` modifier on the internal functions `_processParticipationBNB()`, `_processParticipationUSDT()`, and `_postParticipation()` causes excessive gas consumption. Consider moving reentrancy protection to the public/external functions.

C3. MulaIdoVestor & MulaSeedVestor

Overview

A vesting contract that is intended to be called by the MulaIdo/MulaSeed contract during the sale. Inherits the MulaVestorUtils and MulaTokenUtils contracts.

Issues

C3-01 recordInvestment() must not be called after first unlock

High Resolved

Calling the `recordInvestment()` function after the first successful `withdrawInvestment()` event would cause partially lost funds as the newly vested amount would be split and the first parts become inaccessible.

Recommendation

Restrict the vesting schedule updating and check the first date inside the `recordInvestment()` function.

C3-02 Access to recordInvestment() function

High Resolved

The owner can set EOA as an operator, call `recordInvestment()` and then withdraw all funds.

Recommendation

The function `recordInvestment()` should transfer tokens to the contract from a sender.

C3-03 Contract balance may be insufficient

Medium Resolved

`recordInvestment()` must ensure that the contract has enough tokens to properly record the modification by either comparing the current balance with previous recorded total investment, or transferring the needed tokens from the sale contact via `transferFrom()`. Otherwise users may face the locked funds problem.

C3-04 Code style & gas optimizations

Low Resolved

The event `LogVestingWithdrawal()` could use an indexed parameter for addresses.

In the `getVestingDetails()` function, the `uint256 vestStage` parameter should be provided as `VestingStages vestStage`, which would significantly simplify the code.

A typo in L92 'stagee'.

C3-05 Unsafe math

Low Resolved

The `recordInvestment()` function uses simple addition. We recommend uniforming all mathematical operations to either safe by pragma >0.8.0 or safe by using the imported library.

C4. MulaTokenUtils

Overview

Support contract that implements a whitelist of operators.

Issues

C4-01 Owner exaggerated rights

High Resolved

The owner is able to toggle the `released` boolean variable resulting in toggling the whitelist application to the transfers.

```
function updateRelease() onlyOwner() public {
  released = !released;
}
```

Recommendation

Start of public transfers should be irreversible.

C4-02 Whitelisting is one-way

Medium Resolved

The owner is able to add a wrong address to whitelist, but has no ability to revert this action, causing a sale malfunction.

```
function whitelistOperator(address _operator) onlyOwner() public {
  operators[_operator] = true;
}
```

Recommendation

Consider adding the opposite resulting function or modify the `whitelistOperator()`, e.g.:

```
function whitelistOperator(address _operator, bool _status) onlyOwner() public {
  operators[_operator] = _status;
  emit WhitelistUpdated(_operator, _status);
}
```

C4-03 Typos

● Info ✓ Resolved

L25 'lsit', L37 'isRealease'.

C4-04 Code style & gas optimizations

● Info Partially fixed

The MulaTokenUtils contract could be declared as `abstract`, getting rid of an empty constructor.

Public functions could be declared `external` in order to save gas on calling them.

Require statements in the `view` functions are unnecessary, see `isRealease()`. If any authorization is required, such functions should return zeroes.

No explicit visibilty for `operators[]` mapping is specified.

C5. MulaSaleUtils

Overview

Support contract that holds the sale parameters and handles the Chainlink-like oracle price feed.

Issues

C5-01 Oracle address manipulation

High Resolved

Owner can set oracle address to non-valid address and price calculaction will be broken.

Recommendation

Oracle address should be immutable.

C5-02 Division before multiplication

Low Resolved

In general, we recommend reducing the division error by performing multiplications prior to divisions (minding the possible overflow).

```
    function convertUsdToBNB(uint256 usd,uint80 _roundId) public view returns (uint256){
        uint256 bnbPrice = uint256(getBNBUSDPrice(_roundId));
        return  (usd.div(bnbPrice)).mul(_crossDecimal);
}
```

C5-03 Code style & gas optimizations

Low Resolved

The MulaSaleUtils contract should be declared as `abstract`, getting rid of an empty constructor.

No explicit visibility specified for the variables `_startTime`, `_endTime`, `investorMinContribution`, `investorMaxContribution`, and `_crossDecimal`. Setting them public where needed would

simplify the code.

No need for using require statements in view functions (see `isOpen()`). Such functions should return zeroes/false if the requirements aren't met.

Multiple reads from storage in the `_MulaReceiving()` function: the `_crossDecimal` variable could be read only once to a local variable. The same goes for the `stage` variable in the `isOpen()` function.

Safecast `int -> uint` could be used in the `_MulaReceiving()` function.

Lack of error messages in require statement in the `getBNBUSDPrice()` function.

The `calculatePercent()` function performs excessive calculations: `*100/10000` should be reduced to `/100`.

Variables `_startTime`, `_wallet`, `_USDTContract` can be made immutable. The varialbe `_crossDecimal` can be made constant.

C5-04 Price factor

Low Acknowledged

The `_crossDecimal` price factor should be calculated from the AggregatorV3.decimals value. Changing the oracle price feeds could cause wrong price calculation.

C6. MulaVestorUtils

Overview

Support contract that records the sale participants' addresses and vesting schedule. Inherits the MulaTokenUtils contract.

Issues

C6-01 Token address must be immutable

High Resolved

The owner is able to update vested token address. Loosing the control over the owner account may cause locked funds of users.

```
function updateTokenAddress(IERC20 token) public onlyOwner() returns (bool){
  _token = token;
  return true;
}
```

Recommendation

We recommend avoiding changing the crucial token contract addresses.

C6-02 Missing authorization

High Resolved

The `setVestingDates()` function has public access and doesn't perform any authorization checks. Anyone is able to update the global vesting schedule start.

```
function setVestingDates(uint256 firstListingDate) public {
      provisionDates[VestingStages.TGE] =firstListingDate;
      provisionDates[VestingStages.M1]  =firstListingDate + 30 days;
      provisionDates[VestingStages.M2]  =firstListingDate + (2 *30 days);
      provisionDates[VestingStages.M3]  =firstListingDate + (3 *30 days);
      provisionDates[VestingStages.M4]  =firstListingDate + (4 *30 days);
      provisionDates[VestingStages.M6]  =firstListingDate + (6 *30 days);
      provisionDates[VestingStages.M12] =firstListingDate + (12 *30 days);
}
```

Recommendation

Consider adding a special role of the authorized caller for this function, or add an onlyOwner modifier and implement the caller function in sale contracts.

C6-03 Unsafe math is used

Low Resolved

We recommend using SafeMath with pragma <0.8.0 unless corresponding checks are performed.

```
  function calculatePercent(uint numerator, uint denominator) internal  pure returns
(uint256){
    return (denominator * (numerator * 100) ) /10000;
  }
```

C6-04 Code style & gas optimizations

Low Resolved

No explicit visibility is specified for `provisionDates[]` mapping.

MulaVestorUtils contract could be declared as `abstract`, getting rid of an empty constructor.

In the `getVestingDates()` function, the `uint256 vestStage` parameter should be provided as `VestingStages vestStage`, which would significantly simplify the code.

The `isInvestor()` funciton is redundant as the `investors[]` mapping has public visibility with its own getter function.

The `calculatePercent()` function performs excessive calculations: `*100/10000` should be reduced to `/100`.

C7. Multiple contracts

Overview

Following issues are related to multiple contracts.

Issues

C7-01 Too wide pragma range

Medium Partially fixed

All the reviewed contracts are designed with extremely wide admissible compiler versions.

```
pragma solidity >=0.4.22 <0.9.0;
```

Recommendation

We strongly suggest narrowing the pragma version to the ones that were used for testing. Solidity releases' features and bugfixes should be taken into account.

C7-02 Lack of events

Info Partially fixed

General lack of custom events for functions with external or public visibility.

We recommend to add such events for every major parameter-changing and/or user-interacting function.

5. Conclusion

6 high and 5 medium severity issues were found. All high and 4 medium severity issues were fixed with the update, while 1 medium was fixed partially.

The reviewed contracts are highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

We recommend adding unit and functional tests for the project.

This audit includes recommendations on improving the code and preventing potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY