QuillAudits

Audit Report
April, 2021

m!

Contents

Scope of Audit

The scope of this audit was to analyze and document Mixsome smart contract codebase for quality, security, and correctness.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Gas Consumption
In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	0	0
Closed	1	5	2	0

Introduction

During the period of **April 14th, 2021 to April 17th, 2021** - Quillhash Team performed a security audit for **Mixsome** smart contracts.
The code for the audit was taken from following the official Gitlab link:
https://gitlab.com/fedok_dl/sol-101/-/tree/master/contracts

Commit Hash:

- Root.sol - fe38c4b33b1529e119f347d351242caf0446c037
- Token.sol - 2120081e8c9c25f4d0a0b86d28a5c1036a55dff3

Issues Found – Code Review / Manual Testing

High severity issues

1. State Variable is updated after External Call. Violation of Check_Effects_Interaction Pattern.
   Line no - 1045

Status: Not Considered

Description:
The **loadGroupInfo** calls the **_transferTokensOnLoad** function at Line 47, which includes an external call within its function body(Line 103).

However, some of the state variables of the contract are being updated after this external call has been made. This violates the Check_Effects_Interaction Pattern.

Although the external call is being made to the token contract itself, it is not considered a better practice to modify state variables after the external calls. This could lead to a potential re-entrancy scenario.

Recommendation:
State Variables must be updated before making any external call. Check_Effects_Interaction Pattern should be followed.

2. **Loops are extremely costly**
   Line no: 61,79,90,101,186

Status: Not Considered.

Description:
Every loop in the contract includes state variables like .length of a non-memory array, in the condition of the for loops.
As a result, these state variables consume a lot more extra gas for every iteration of the for loop.
The following functions include such loops at the above-mentioned lines:

- **_updateTotalGroupBalance**
- **_checkTotalPercentSumInGroup**
- **_setupBaseBalance**
- **_transferTokensOnLoad**
- **_getSenderIndexInGroup**

Recommendation:
Its quite effective to use a local variable instead of a state variable like .length in a loop.
For instance,

```
uint256 local_variable = _groupInfo.addresses.length;
 for (uint256 i = 0; i < local_variable; i++) {
          if (_groupInfo.addresses[i] == msg.sender) {
              _isAddressExistInGroup = true;
              _senderIndex = i;
              break;
          }
      }
```

3. **Return Value of External call is never used**
   Line no: 97

Status: Not Considered.

Description:
The return value of the externals made in the following functions are never used:

- **_transferTokensOnLoad**

Recommendation:
Effective use of all return values must be ensured within the contract.

4. Division is being performed before Multiplication.

Status: CLOSED.

Description:
During the automated testing of the Root.sol contract, it was found that the _getWithdrawPercent function performs a multiplication on the result of a Division.

Integer Divisions in Solidity might truncate.

```
155         uint256 _daysFromDeploy = (block.timestamp).sub(_deployTime).div(24 * 3600).mod(30);
156         uint256 _daysWithdrawPercent = _dayWithdrawPercent.mul(_daysFromDeploy);
```

Recommendation:
Solidity doesn't encourage arithmetic operations that involve division before multiplication. Therefore the above-mentioned function should be checked once and redesigned if they do not lead to expected results.

5. Redundant Variable Initialization
Line no - 140

Status: Not Considered.

Description:
The **_getWithdrawPercent()** function initializes a local variable **_deployTime** with the already available state variable **deployTime**(Line 30).

Since the state variable **deployTime**(Line 30) has already been initialized in the constructor of the Root Contract(Line 34), it can be used directly in the computations performed in the **_getWithdrawPercent** function.

```
137     function _getWithdrawPercent(GroupInfo memory _groupInfo)
138         uint256 _index = 0;
139         uint256 _timePerIndex = 30 days;
140         uint256 _deployTime = deployTime;
141
```

Recommendation:
Avoid redundant initialization of variables.

6. **Strict Equality is being used in the IF statement**
   Line no: 122-124

Status: CLOSED.

Description:
The **_checkTotalGroupsBalance** function includes a STRICT EQUALITY check in the **require** statement.

Since the strict equality check is in the require statement it does demand the **totalGroupsBalance** state variable to be precisely equal to **127500000000000000000000000**, and if it does not, the function will revert back. This could lead to a very unexpected scenario where the **loadGroupInfo** function won't get executed since it calls the **_checkTotalGroupsBalance** within its function body.

Is this Intended?

Recommendation:
It is not considered a better practice in Solidity to implement a Strict Equality check in the If or **require** statements.

If the above-mentioned logic is not intended, then the **require** statement should be modified. Instead of using a **strict equality sign(==), greater than or equal to(>=)** could be used.

Low level severity issues

7. **Explicit visibility declaration is missing**
   Line no - 25, 26,27

Status: CLOSED

Description:
The following element in the contract have not been assigned any visibility keyword explicitly:

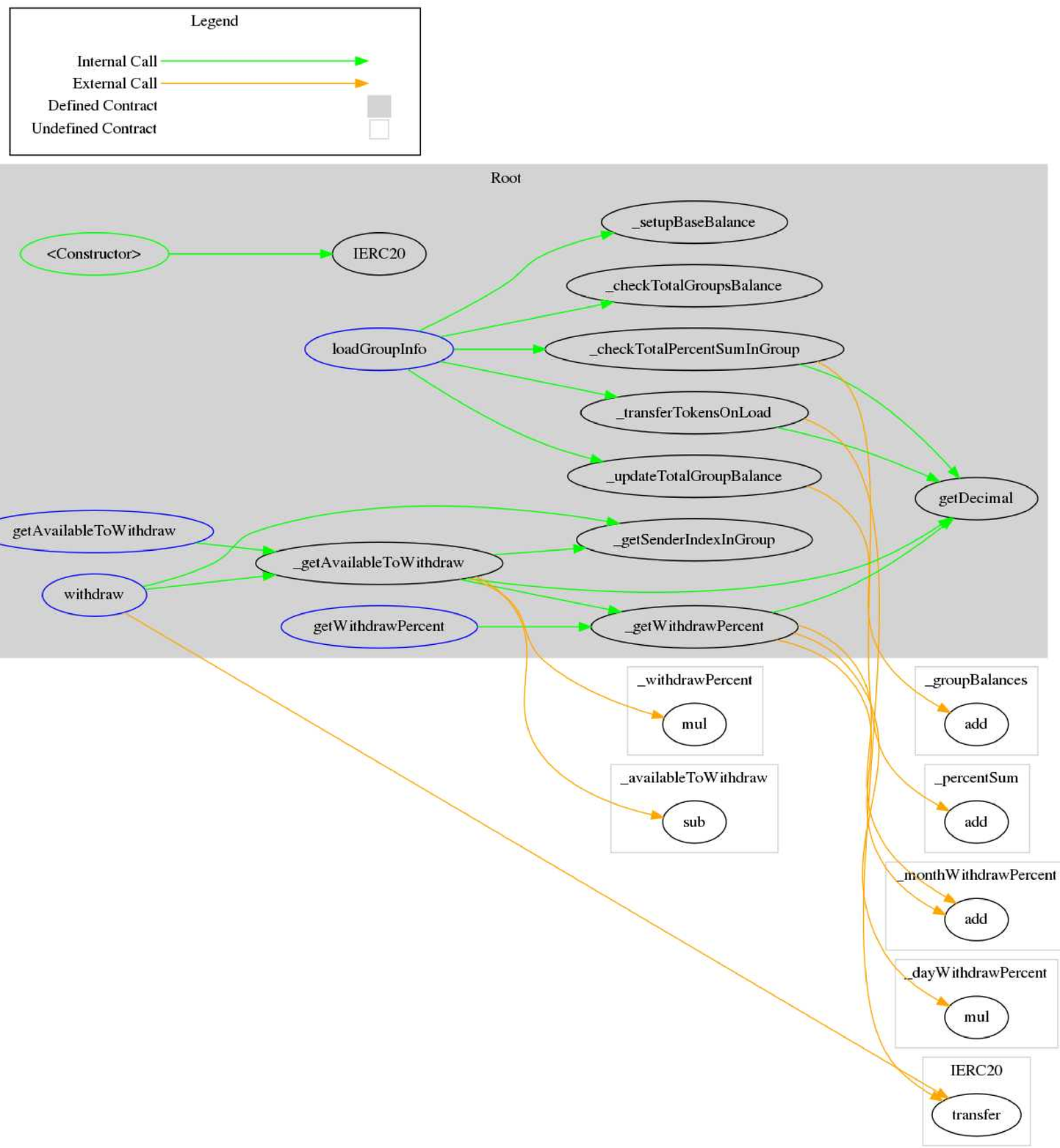
- **groupToInfo** mapping at Line 25
- **groupToInfoCount** at Line 26
- **totalGroupsBalance** at Line 27

Recommendation:
Visibility specifiers should be assigned explicitly in order to avoid ambiguity.

8. Coding Style Issues

Status: CLOSED

Coding style issues influence code readability and, in some cases, may lead to bugs in the future. Smart Contracts have a naming convention, indentation, and code layout issues. It's recommended to use Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.



Closing Summary

Overall, smart contracts are very well written and adhere to guidelines. During the process of the final audit, it was found that some issues from the initial audit report have not been considered and implemented. However, No high severity issues or instances of Re-entrancy or Back-Door Entry were found in the contract.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **Mixsome platform**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Mixsome Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

m !

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com