Introduction

Ritesh from ChainBlocks was engaged to perform security audit for Emrify smart contracts. The audited code was taken from One Public github repositories comprising of 4 smart contracts:

`https://github.com/emrifylabs/EmrifySC/`

The links for the smart contracts is

- https://github.com/emrifylabs/EmrifySC/blob/master/SmartContracts/ERC20.sol
- https://github.com/emrifylabs/EmrifySC/blob/master/SmartContracts/ERC20HIT.sol
- https://github.com/emrifylabs/EmrifySC/blob/master/SmartContracts/Hodler.sol
- https://github.com/emrifylabs/EmrifySC/blob/master/SmartContracts/library.sol

The Audit has been conducted on b5d64f26 commit in this repository on 25th August 2018. Subsequently code had been changed and committed to the repo on 30th September 2018 based on the review comments. The commit number for this change is fb0c39fa.

[**Update]**

The review on final code was conducted on 21st October 2018 on 887a70a and the contracts are suitable for deployment. The review has been conducted based on the best available knowledge and information and customer is satisfied with the reviews.

Disclaimer

The audit makes no statement or warranties about utility of the code, safety of the code, suitability of the business model, regulatory, regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status.

The report contains information about both prior and after correction of smart contracts.

1.1 Overview of the Emrify platform

Emrify is a healthcare related platform built on Ethereum. It allows users to purchase HITT tokens.

The Emrify platform collects Ethers from its users in return of HITT tokens. It provides bonus and bounties to account holders holding its token for 3, 6, 9 and 12 months respectively. It is based on standard ERC20 token implementation.

1.2 Scope of audit

In the audit we reviewed the smart contracts that implement the token mechanism, which includes an implementation for all ERC20 token operations as well as calculation and transfer of bonus token to holders for long term.

The following files were reviewed:

1. `ERC20.sol`
2. `ERC20HIT.sol`
3. `Hodler.sol`
4. `library.sol`

2. Summary of Findings

Overall, the code is clearly written, and demonstrates effective use of abstraction, separation of concerns, and modularity. Emrify development team demonstrated high technical capabilities, both in the design of the architecture and in its implementation.

We found one critical issue and few non-major additional issues that require the attention of the Emrify team. Given the subjective nature of some assessments, it will be up to the Emrify team to decide whether any changes should be made.

2.1 Major Issues

2.1.1 No check is made is an address belongs to a contract or to an individual

- Likelihood: *High*
- Impact: *high*

There is no check to ascertain whether the transfer is made to a contract account or an individual owned account.

This has been fixed in fb0c39fa commit.

2.1.2 Race Condition in multiple functions related to shared variables

- Likelihood: *Medium*
- Impact: *high*

Multiple shared variables are used across functions including ones that transfer tokens to other accounts. There is a possibility that read and write operations might be happening in them at same time for same user. Functions addHodlerStake and invalidate should be modifier from this issue perspective.

This has been fixed in fb0c39fa commit.

2.1.3 No Check on dynamic address array in functions

- Likelihood: *high*
- Impact: *high*

claimHodlRewardsForMultipleAddresses function accepts a dynamic array of address from anyone. It does not check for who is invoking this function neither has any other checks. It is open to be called by anyone external. It is possible that this function introduces attacks on the contract.

2.1.4 A bug in transfer prevents recipient fees from being collected

- Likelihood: *high*
- Impact: *medium*

This has been fixed in fb0c39fa commit.

2.2 Other Issues

2.2.1 In general, post transaction, a validation of existing global state is not conducted

- Likelihood: *Low*
- Impact: *Low*

This has been fixed in fb0c39fa commit.

2.2.2 Calculations in constructors can be determined as constants

- Likelihood: *Low*
- Impact: *Low*

2.2.3 Data Type inconsistencies at few places

- Likelihood: *Low*

- Impact: *Low*

batchTransfer and saleDistributionMultiAddress function is accepted uint256 number of address and using its length for iteration. However, the for loop is using uint8 datatype

This has been fixed in fb0c39fa commit.

2.2.4 Few places got missed to implement safe maths to avoid overflow attack especially in for loops.

- Likelihood: *Low*
- Impact: *Low*

finalizeHodler, batchTransfer, saleDistributionMultiAddress, HITT constructor are evidences for this issue.

This has been fixed in fb0c39fa commit. Still couple of places need changes.

2.3 Implementation Recommendations

2.3.1 Check for Address type whether is belongs to a contract or individual owned account.

2.3.2 Implement Mutex to avoid race conditions arising because of usage of shared global variable between multiple functions.

Functions addHodlerStake

~~2.3.3 Check for address length before using it.~~

2.3.4 Convert public variables to private or internal if possible

2.3.5 Remove cyclic dependency between contracts by replacing some of the functions from ERC20 token contract to Hodler contract.

Function saleDistributionMultiAddress and batchTransfer can be moved to Hodler contract.

2.3.6 Check for incoming arguments in all functions including constructor using require statements.

2.3.7 Convert public variables to private or internal
2.3.8 Where ever possible check for upper and lower array boundaries.

2.3.9 Use safe maths for any calculations like addition, substaction, multiplication and division.

2.3.10 Assert after every state change function to ascertain if the global state is in consistent state.

2.3.11 Delete should be implemented as soft delete in a contract unless the same is captured using events and stored elsewhere.

2.3.12 Consistent use of Modifiers. If modifiers are defined use them consistency to determine the intent of the function.

3. Good aspects of the contract

Some of the good implementation are mentioned here although they are not exhaustive.

a. Valid use of Pragma
b. Right way to transfer ether between accounts – using the transfer method.
c. Correct use of require statements
d. Valid use of check-effect and interact pattern
e. Valid use of modifiers
f. Each function returns a type to denote success or failure
g. Usage of Maths library to avoid overflow and underflow issues
h. Usage of latest features of Solidity like constructor
i. No use of tx.origin
j. Input validation is quite robust