Icecream 2018 ICO Audit Report

Introduction

In this Smart Contract audit, we'll cover the following topics:

1. Disclaimer
2. Overview of the audit and nice features
3. Attack made to the contract
4. Critical vulnerabilities found in the contract
5. Medium vulnerabilities found in the contract
6. Low severity vulnerabilities found
7. Line by line comments
8. Summary of the audit

1. Disclaimer

The audit makes no statements or warrantees 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 audit documentation is for discussion purposes only.

2. Overview

The project has four files, the ERC20.sol file which is defining the ERC20 token's interface comprising of its functions and events, it is 14 lines of code and enough documented. The Migrations.sol file is 23 lines of solidity code file and is well written but it needs to be commented to be easy for a new comer to understand. RecToken.sol is a token declaration file, it inherits PausibleToken and BurnableToken from the Open Zeppelin and the Rec Token is finally declared here. The file is easy to understand for a developer but might put hurdles to a non-tech person. RecTokenCrowdsale.sol is a crowdsale declaration file, it is well written, and the code modularity is appreciated. This file is well documented as well.

The whole code overall is written according to Open Zeppelin standards and practices.

The purpose of the smart contract is to create an ERC 20 token symbolled REC and its crowd sale to distribute the tokens according to the TOKEN PLAN.docx file provided.

Nice Features

The contract provides a good suite of functionality that will be useful for the entire contract:

1. The contract is well written with good use of data structures
2. The efficient code approaches are being used

3. Attacks made to the contract

To check for the security of the contract, we tested several attacks to make sure that the contract is secure and follows best practices.

Reentrancy attack

This attack consists on recursively calling the call.value() *method in a ERC20 token to extract the ether stored on the contract if the user is not updating the* balance *of the sender before sending the ether.*

When you call a function to send ether to a contract, you can use the fallback function to execute again that function until the ether of the contract is extracted.

Because this contract uses transfer() instead of call.value() , there's no risk of reentrancy attacks since the transfer function only allows to use 23.000 gas which you can only use for an event to log data and throws on failure.

That way you're unable to recursively call again the sender function thus avoiding the reentrancy attack.

The transfer function is called only when distributing the funds to the wallet which happens once per token purchase, so there shouldn't be any problem with reentrancy attacks.

Over and under flows

An overflow happens when the limit of the type varibale `uint256` *, 2**256, is exceeded. What happens is that the value resets to zero instead of incrementing more.*

For instance, if I want to assign a value to a uint bigger than 2**256 it will simple go to 0 — this is dangerous. On the other hand, an underflow happens when you try to substract 0 minus a number bigger than 0. For example, if you substract 0 -1 the result will be = 2**256 instead of -1. This is quite dangerous when dealing with ether.

However, this contract is based on Open Zeppelin which itself is using SafeMath's functions for add(), mul(), div() and sub(), which actually ensures we don't encounter overflow or underflow conditions, so the contract is safe for this attack.

Replay attack

The replay attack consists on making a transaction on one blockchain like the original Ethereum's blockchain and then repeating it on another blockchain like the Ethereum's classic blockchain.

The ether is transferred like a normal transaction from a blockchain to another. Though it's no longer a problem because since the version 1.5.3 of Geth and 1.4.4 of Parity both implement the attack protection EIP 155 by Vitalik Buterin:

https://github.com/ethereum/EIPs/blob/master/EIPS/eip-155.md

So, the people that will use the contract depend on their own ability to be updated with those programs to keep themselves secure.

Short address attack

This attack affects ERC20 tokens, was discovered by the Golem team and consists of the following:

- A user creates an ethereum wallet with a traling 0, which is not hard because it's only a digit. For instance: 0xiofa8d97756as7df5sd8f75g8675ds8gsdg0
- Then he buys tokens by removing the last zero:

Buy 1000 tokens from account 0xiofa8d97756as7df5sd8f75g8675ds8gsdg

- If the token contract has enought amount of tokens and the buy function doesn't check the length of the address of the sender, the Ethereum's virtual machine will just add zeroes to the transaction until the address is complete.
- The virtual machine will return 256000 for each 1000 tokens bought. This is a bug of the virtual machine that's yet not fixed so whenever you want to buy tokens make sure to check the length of the address.

The contract is vulnerable to this attack since its ERC20 based and as far as I checked it is not checking the length of the address. So, a suggestion is to check the length of the input address, check solution here: https://medium.com/huzzle/ico-smart-contract-vulnerability-short-address-attack-31ac9177eb6b

You can read more about the attack here: http://vessenes.com/the-erc20-short-address-attack-explained/

4. Critical vulnerabilities found in the contract

There are no critical issues found in the contract.

5. Medium vulnerabilities found in the contract

The medium vulnerability found is the short address length attack.

6. Low severity vulnerabilities found

A recommendation is that in definitionStage() function inside RecTokenCrowdsale.sol file, you should not use direct '+' and should use SafeMath's function add() to do this.

7. Line by Line comments

Line 1 | file RecTokenCrowdsale.sol, RecToken.sol, Migrations.sol, ERC20.sol: You're specifiying a pragma version with the caret symbol (^) up front which tells the compiler to use any version of solidity

bigger than 0.4.23. This is not a good practice since there could be major changes between versions that would make your code unstable. That's why I recommend setting a fixed version without the caret like 0.4.23.

Line 144 | file Crowdsale.sol: As written in the Token Plan.docx *"The tokens will be delivered to the participants of the Token Sale after the Token Sale End Date. Transferability of the tokens is governed by the applicable vesting period, if any."*

The tokens are being delivered to each investor immediately as per they send it, so my recommendation is to counter check if this requirement is needed or not and then implement or not implement accordingly.

8. Summary of the audit

Overall the code is well commented and clear on what it's supposed to do for each function, have a look to the second line by line comment and counter check your requirement if it is needed or not.

This is a secure contract that will store safely the funds while it's working.