Divicoin Token Contract: Review

Mikhail Vladimirov and Dmitry Khovratovich

4th September, 2017

This document describes issues found in Divicoin during code review performed by ABDK Consulting.

1. Introduction

We were asked to audit the Divicoin Token Contract at the commit 71f15bfa. The contract is contained in Divi-Crowdsale-v0_1_05.sol.

We got no additional documentation on the contracts so reviewed them as is.

2. Divi-Crowdsale

In this section we describe issues related to the token contract defined in Divi-Crowdsale-v0_1_05.sol.

2.1 ERC-20 Compliance Issues

The `approve` function throws if the current allowance is not 0, which contradicts the ERC-20 requirements and is thus incompatible with pre-existing token processing software (line 423).

2.2 Documentation Issues

1. Names STARTDATE and ENDDATE are confusing, the variable actually stores start time, not date.
2. The documentation on multisigTokens (line 326) implies that the Foundation gets 20% of newly created tokens whereas the operation below means that the Foundation gets 500% or ⅙ of all token supply.

2.3 Arithmetic Overflow Issues

This section lists issues of token smart contract related to arithmetic overflows.

1. The addition (line 41) and multiplication (line 77) operations rely on how overflow works in Solidity, but this behavior is not documented and may change in future. We recommend to redevelop the functions so that the overflow never actually happens.
2. Overflow is possible at line 330.

2.4 Subefficient Code

This section lists subefficient code patterns found in token smart contract.

1. As the division method does not implement any protection, it can be replaced just by regular / operation (line 49).Amount of information being logged at line 350 is redundant. For example, value of `tokens` parameter may be calculated from values of `ethers` and `buyPrice`. Also, name of `ethers` parameter is confusing, because it actually contains value in Wei.
2. Value of `balances[msg.sender]` is read second time at line 374, which is suboptimal.

2.5 Redundant Functionality

1. The “short address” protection (line 154) looks redundant. There are too many ways to call a smart contract incorrectly (for example, confusing the order of parameters), of which the “short address” issue is the most known and thus usually fixed.
2. The `buyPriceAt` and `buyPrice` functions are not used after the contract is deployed, and thus the price will not depend on the time.

2.6 Double Approve Issue

The ERC-20 API is vulnerable to the double-approve attack, where an allowance recipient is able to race the spender in order to benefit from two consecutive approvals. The contract protects from it by requiring the allowance to be 0 when approve function is called. As this breaks backward compatibility, we recommend introducing another approval function which would have a built-in protection. Such function can be defined as `approve(address _spender, uint _oldValue, uint _newValue )` taking assumed allowance value as the second argument. If actual allowance differs from an assumed one, this method just returns false.

2.7 Major Flaws

1. The price calculated in the contract will be always zero. The reason is that the value of `_buyPrice` storage variable will be calculated and assigned only once (line 196), before the constructor is called, and most probably this value will be zero. It is then used in all price calculations.
2. The `multisigTokens` variable is assigned later (line 330) than it is first used (line 308). Thus it will be equal to 0 in operations before line 330.
3. The `onlyPayloadSize` modifier is designed for two inputs where three inputs are used in line 388.
4. The `transferFrom` method functionality looks broken:

a. `balances[msg.sender]` should be `balances[_from]` at line 402;
b. `balances[msg.sender]` should be `balances[_to]` at line 404;
c. `balances[msg.sender]` should be `allowed[_from][msg.sender]` at line 406;

2.8 Other Issues

This section lists stylistic and other minor issues found in the token smart contract.

1. Name collision warning for `totalSupply` will be reported at (line 7) .
2. In the `onlyPayloadSize` modifier (line 155) the `require` operation should probably use `==` rather than `>=`.
3. In Solidity, `now` is an alias for `block.timestamp`. Accessing block meta data from constant functions (line 221) is not a good idea because constant functions may be executed locally, outside any transaction and block.
4. The check for zero tokens (line 322) should be probably moved right after the variable is assigned.
5. It might be cheaper to accumulate funds on contract's balance and then sent to multisig in bulk (line 346).
6. Why to disallow transfers of zero tokens (line 370, 396)? Most token contracts do allow this.
7. The value of `_totalSupply` is always zero at line 164. Should be replaced with the constant or redesigned.

3. Security provisions

We recommend the smart contract designers to claim explicitly security provisions in addition to the intended functionality of the contract. Such provisions should be non-trivial falsifiable claims, i.e. they should declare certain property of the contract for which an attack can be demonstrated if implemented incorrectly. The security provisions serve not only for the purpose of confidence of future users in the contract's behaviour, but also as a platform for a potential bug bounty program. The contract authors are encouraged to offer various bounties binded to some provision being violated.

Here we provide a list of security provisions which are appropriate for these contracts and would be expected by ordinary users.

Name	**Description**	**Current status**
Divicoin		
Constant supply.	Sum of all positive balances is constant	FALSE (section 2.7.4)
Safe multiple approvals.	A token holder can approve another address with some tokens	TRUE

	only if the previous allowance has been fully spent.	
No lost Ether	Every Ether that is sent to the contract is either reverted or is converted to tokens and forwarded to the contract owner.	FALSE (Section 2.7.1)

4. Our Recommendations

Based on our findings, we recommend the following:

1. Fix the price calculation method;
2. Fix the `transferFrom` method;
3. Fix or remove the onlyPayloadSize modifier;
4. Fix overflow issues;
5. Refactor the code to remove subefficient parts.