HighTechBlock

Smart Contract Security Audit

Introduction

We reviewed and audited the ABC token contract.

The version of the audit is based on commit *45f661636de88c76f5312eccd72123052452f7e98* of the public *https://github.com/sample-contract-address*.

Executive Summary

We reviewed the ABC token contract and found 3 critical security problems. Our findings included 3 important, 1 medium and 2 low severity.

Issue	Description	Severity
1	ABC contract, tokenCreationCapOne should be 25000000	Critical
2	Soft cap is hardcoded at 5000 ether, whereas the white paper states $1.5M	Critical
3	Redeeming tokens is not being prevented before the end of the token sale	Critical
4	Owner account access should be protected using a multisig, and each key stored on different hard wallets	Important
5	Status.hasEnded() is only true after the end date (November 30th 8pm GMT), regardless of whether all the funds have been gathered already.	Important
6	No whitelisting process.	Important
7	Proper usage of require() vs assert()	Medium
8	Usage of ERC223 over ERC20	Low
9	Upgrade to Solidity 0.4.17	Low

HighTechBlock

Findings

Critical

#1 ABC contract, tokenCreationCapOne should be 25000000, not 75000000, in order to match the white paper specifications.

#2 Soft cap is hardcoded at 5000 ether, whereas the white paper states $1.5M

#3 Redeeming tokens is not being prevented before the end of the token sale. Upon transferring ether into the smart contract (function () payable), createTokens() gets called, which calls addTokens, which will increase the balances array for the respective sender. This is essentially attributing the tokens to the investors. Also, investors are able to transfer their tokens to other addresses using the transfer and transferFrom functions. This contradicts the white paper requirement that the tokens should only be released at the end of the crowdsale.

Important

#1 Owner account access should be protected using a multisig, and each key stored on different hard wallets. This ensures that if one of the keys gets hacked or lost, the owner would still have access to the account and not risk losing all the funds.

#2 Status.hasEnded() is only true after the end date (November 30th 8pm GMT), regardless of whether all the funds have been gathered already. This makes it impossible to finalize the sale (send the tokens to investors) before the end date, even if the hard cap has been reached.

#3 No whitelisting process. Although the paper doesn't require it, we recommend having a KYC & whitelisting process, so that funds wouldn't be accepted from unregistered investors.

Moderate

#1 *Proper usage of require() vs assert()*

assert() should be used to enforce invariants and checking for states and error conditions which should be unreachable is code logic is correct

require() should be used to validate inputs, contract state, or return values from calls to external contracts.

Another important difference is that after Metropolis, require() will become revert(), which will refund the remaining gas, and return a value, which can be useful for debugging.

Low

#1 Usage of ERC223 over ERC20.

Main benefits and improvements are:

1) Eliminates problem of lost tokens. This happens when users mistakenly use a contract instead of a wallet address when transferring tokens. This leads to tokens being lost. ERC223 allows users to send their tokens to either a wallet or contract.
2) Allows developers to handle incoming token transfers, and reject non-supported tokens
3) Energy savings. Transfer of ERC223 tokens to a contract is a one step process, instead of a 2 step for ERC20. This leads to less gas required for the operation.

#2 Upgrade to Solidity 0.4.17

Known vulnerabilities

1) The methods of StandardToken use the proper protection mechanisms to protect against the short address size attack of ERC20.

2) Usage of SafeMath library and operations.

3) No usage of transaction origin in the contract.

4) No reliance on block timestamps.

5) Proper usage of checks-effects-interactions pattern.

6) Proper usage of function visibility modifiers.

7) Proper handling of fallback functions.

8) Proper protection against reentrancy attacks.

Limitations

Only the Solidity source code of the smart contract was reviewed. The usage, migration, and deployment of the smart contract can also affect the crowdsale.