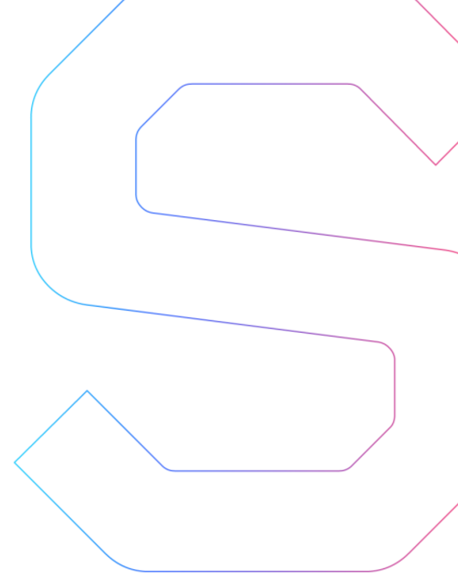

SmartDec

VoiceOfCoins Smart Contracts Security Audit

Abstract

In this report we consider the security of the VoiceOfCoins project. Our task was to find and describe security issues in the smart contracts of the platform. Several issues of different severity level have been found and reported to the developer. All of them were fixed by the developer and are not present on commit f86ecf6.

Procedure

In our analysis we consider VoiceOfCoins whitepaper (VOC Index Fund .pdf, sha1sum 37955f43c3b85811c4691a649dafab7005e13ac3) and smart contracts code (version on commit 98ef76e).
We perform our audit according to the following procedure:

- automated analysis
  - we scan project's smart contracts with our own Solidity static code analyzer SmartCheck
  - we scan project's smart contracts with several publicly available automated Solidity analysis tools such as Remix, Oyente, Securify (beta version since full version was unavailable at the moment this report was made) and Solhint
  - we manually verify (reject or confirm) all the issues found by tools
- manual audit
  - we manually analyze smart contracts for security vulnerabilities
  - we check smart contracts logic
- report
  - we reflect all the gathered information in the report

Disclaimer

The audit does not give any warranties on the security of the code. One audit can not be considered enough. We always recommend proceeding to several independent audits and a public bug bounty program to ensure the security of the smart contracts. Besides, security audit is not an investment advice.

Checked vulnerabilities

We have scanned VoiceOfCoins smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- 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

About The Project

Project Architecture

The project consists of a single source code file VOCTOP25.sol. The code on commit 98ef76e implements:

1. the `owned` contract which is responsible for authorizing addresses using the `onlyOwner` modifier
   The contract was renamed to Owned on commit f86ecf6.
2. the `tokenRecipient` interface which contains the definition of the one `receiveApproval` function, which is designed to handle approving tokens to the address of the contract
   There is no tokenRecipient interface on commit f86ecf6.
3. the `VOCTOP25` contract is the main project contract that implements the token (ERC20 token with some violations — they are described in the report below — and with additional functionality).
   The is no ERC20 standard violations on commit f86ecf6.
4. The project does not contain automatic tests and deploy scripts.
   Automatic tests and deploy scripts are present on commit f86ecf6.

Code Logic

The code implements the logic of the token. The token partly corresponds to the ERC20 standard (the discrepancies are described in the report below).
There are no ERC20 standard violations on commit f86ecf6.

Symbol — VOC25, decimals — 18, name — “Voice Of Coins TOP 25 Index Fund”, totalSupply depends on how many tokens are generated by the `mintToken` function.
In addition to the ERC20 standard functionality of the token, the following features are implemented (the owner of the contract is the address from which the the contract was deployed):

1. The contract owner can include an address in the `frozen` list, so the account will be frozen. Also, the owner of the contract can remove an address from this list. Tokens from frozen addresses can not be transferred. The logic contains a vulnerability, see the report.
   Also tokens can not be transferred to frozen addresses on commit f86ecf6. The vulnerability has been fixed by the developer and is not present on commit f86ecf6.
2. In addition to the standard `approve` function, one can call the `approveAndCall` function, which, in addition to calling the usual `approve`, calls the `receiveApproval` function from the address for which tokens are being approved.
   approveAndCall function is not present on commit f86ecf6.
3. The owner of the contract can burn tokens from any address.
4. The contract owner can create (`mint`) tokens to his/her address.

Automated Analysis

We used several publicly available automated Solidity analysis tools. Here are the combined results of their analysis. All the issues found by tools were manually checked (rejected or confirmed).
There are no confirmed issues on commit f86ecf6.

Tool	Rule	false positives	true positives
SmartCheck	Address hardcoded	2	
	Constant functions		8
	Pragmas version		1
	Reentrancy external call		1
	Should be pure but is not	5	
	Should be view but is not	10	
	Unchecked math	6	
	Visibility		8
Total SmartCheck		23	18
Solhint	Comma must be separated from next element by space		2
	Compiler version must be fixed		1
	Contract name must be in CamelCase		2
	Definition must be surrounded with two blank line indent		3
	Definitions inside contract / library must be separated by one line		2
	Event and function names must be different	2	
	Explicitly mark visibility of state		8
	Visibility modifier must be first in list of modifiers		5
Total Solhint		2	23
Remix	Gas requirement of function unknown or not constant	3	1
Total Remix		3	1
Overall Total		28	42

Securify* — beta version, full version is unavailable.
Securify, Oyente — these tools do not support 0.4.18 compiler version; the specified version was changed in the code to 0.4.16 and 0.4.17 respectively for these tools.
Cases when these issues lead to actual bugs or vulnerabilities are described in the next section.

Manual Analysis

Contracts were completely manually analyzed, their logic was checked. Besides, the results of automated analysis were manually verified. All confirmed issues are described below.

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

Frozen funds logic

There is a bug in frozen funds logic. When one calls the `transferFrom` function, he/she is spending not his/her funds. However, the check at line 129 that is called inside the `_transfer` function (which is called inside the `transferFrom`):

```
require(!_frozenAccount[msg.sender]);
```

checks whether the `msg.sender`'s account is frozen, not the `_from` account. We recommend addressing the issue and clarifying what is meant by “freeze” — freezing the account (forbidding it to perform any actions) or the funds.
In the current implementation, the following situation is possible. Let us say, User1 account is frozen. He/she calls `approve` to another address. User2 transfers the approved funds. Thus, the frozen logic is broken. The check

```
require(!_frozenAccount[_from]);
```

might help in this particular situation; however, we recommend clarifying the logic.
The issue has been fixed by the developer and is not present on commit f86ecf6.

Medium severity issues

Medium issues can influence smart contracts operation in current implementation. We highly recommend addressing them.

No tests

The provided code does not contain tests. Testing is crucial for code security. Manual testing should not be considered enough. We highly recommend not only to cover the code with tests but also to make sure that the coverage is sufficient.
The issue has been fixed by the developer and is not present on commit f86ecf6.

ERC20 standard violation

There are several ERC20 standard violations:

- `transfer` does not return `bool` (it does not return anything at all)
- no `Approve` event, so `approve` does not fire `Approval` event
- in the `Transfer` event, the names of the parameters are incorrect (they should be specified with underscores, so that the search for standard names will be possible)

The issue has been fixed by the developer and is not present on commit f86ecf6.

ERC20 approve issue

There is ERC20 approve issue (VOCTOP25.sol, line 196). We recommend instructing users not to use `approve` directly and to use `increaseApproval`/`decreaseApproval`

functions (or to change the approved amount to 0, wait for the transaction to be mined, and then to change the approved amount to the desired value) instead - link (changing the approved amount from a nonzero value to another nonzero value allows a double spending with a front-running attack).
Functions `increaseApproval`/`decreaseApproval` are present on commit f86ecf6.

Low severity issues

Redundant code

- Line 149: the condition in
  ```
  assert(_balanceOf[_from] + _balanceOf[_to] == previousBalances);
  ```
  will always be met. It follows from the logic described above, so `assert` is redundant and will just spend extra gas each time.
- Line 238: redundant check
  ```
  require(_totalSupply >= _value);
  ```
  Total supply will always be greater than or equal to the balance of the owner (because they are increased together by the same amount in `mint`), and the comparison of the value and balance of the owner is on the line 235.

The issue has been fixed by the developer and is not present on commit f86ecf6.

approve logic

The `approve` function (line 196) does not allow to make `_allowance` more than the number of tokens the `msg.sender` has. However, there is a check for not spending more funds than one has in the `_transfer` function (through which all transfers pass). Thus, the check in `approve` is redundant. Besides, it deprives one from approving tokens for the future (when he/she will has more tokens).
The issue has been fixed by the developer and is not present on commit f86ecf6.

Another contract is called

There is a call of another contract at line 218. We recommend to warn users to carefully check what contract will be called.
The issue has been fixed by the developer and is not present on commit f86ecf6.

Code Style

The code violates the Style Guide for Solidity.
The issue has been fixed by the developer and is not present on commit f86ecf6.

Implicit visibility level

In many places in the code there are variables with implicit visibility level. We recommend specifying visibility level explicitly and correctly.
The issue has been fixed by the developer and is not present on commit f86ecf6.

Constant functions

We recommend to use `view` instead of `constant`, which will be deprecated for functions. If a function is not supposed to modify the state or read from state, consider declaring it as `pure`.
The issue has been fixed by the developer and is not present on commit f86ecf6.

Conclusion

In this report we have considered the security of VoiceOfCoins smart contracts. We performed our audit according to the procedure described above.
Several issues of different severity level have been found and reported to the developer. All of them were fixed by the developer and are not present on commit f86ecf6.

This analysis was performed by SmartDec

COO Sergey Pavlin

December 13, 2017