Smart Contract Audit

STARAMBA.Token

August 05, 2018 | v1.1

Introduction

We reviewed and audited the smart contract of the STARAMBA Token. The audit is based on commit bbeff0a816941dd43b2b0595a24e6917befd5955 of the staramba/com.staramba.token.STT GitHub repository [1].

Only the files that implement the contract itself (StarambaToken.sol, StandardToken.sol, SafeMath.sol and RelocationToken.sol) have been reviewed. We omitted any other file in the repository.

We took the “STARAMBA.token Whitepaper - Version 1.0 - 23rd May 2018“ – especially chapter 4.2 “Details of Token design” –as the specification of the contract.

[1] https://github.com/staramba/com.staramba.token.STT

Limitations

We just reviewed and audited the Solidity source code of the smart contract. We did neither review the low-level assembly code that is created by the Solidity compiler nor the deployed contract.
We only audited the smart contract code and did not evaluate the STARAMBA project in any other matters.

Executive Summary

We did not find any critical vulnerabilities in the reviewed version of the STARAMBA.Token Smart Contract. The contract itself follows the coding standards for smart contracts and the overall complexity of the contract is low.

In addition to the normal ERC20 functionality, the contract allows the contract owner to deliver up to 1,200,000,000 tokens. The token buy itself is handled offline.

The transfer of tokens can be paused and resumed by the contract owner which prohibits any calls to transfer() and transferFrom().

In order to migrate to another contract, the contract owner can enable a relocation mode. This allows any token holder to deliver his tokens to a new contract

We suggest to announce an open bug bounty program and to check whether the token symbol should be changed to a more unique symbol.

Findings

Finding	Description	Severity	Resolved
#1	No open bug bounty program	LOW	
#2	Token symbol used by other tokens	INFO	

Extended Review

General

Check Solidity version

The contract should use the latest stable Solidity version to ensure that the latest security patches are applied when compiling the contract.

✓ The contract uses the latest Solidity version 0.4.24.

Check for open bug bounty program

Announcing a bug bounty program is good security practice to incentivize security researchers to further review the contract.

● The project did not announce a bug bounty program for the token contract.

Finding #1: No open bug bounty program
We recommend to announce an open bug bounty program.

Token name and symbol should be available

To avoid confusions, the name and symbol of the token should not be taken by another ERC20 token, yet. Or the existing tokens should not be actively traded on exchange platform.

● According to etherscan [1] there are currently other tokens with the same symbol (STT). The "Synthestech Token" [2] and the "Startup Token" [3] have multiple token holders. The other tokens don't seem to be actively used or traded.

[1] https://etherscan.io/tokens?q=stt
[2] https://etherscan.io/token/0x3e9ba4be6affa324f188b10962317239e32ce7ad
[3] https://etherscan.io/token/0xca15d092b29a1ef9005bb417463d26797842b0a3

Finding #2: Token symbol used by other tokens

Using the same symbol as other tokens can complicate the listing on exchanges if the symbol is already used. We recommend to change the token symbol to a more unique identifier.

Use MultiSig account as contract owner

> *Critical operations within the contract (e.g. retrieval of the funds) should not be authorized by a single person. It is advised to use a multi signature scheme to increase the security of the contract.*

✓ The contract itself implements a four-eye principle for critical operations. Those operations are only executed after both defined admins approved the operation.

Known Vulnerabilities

Mitigate against approve race condition

> *Changing the allowance with the approve function brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering.*
>
> *There is no actual mitigation against this flaw since the flaw is part of the ERC20 token standard itself. However, a comment in the source code should describe the safe use of this function.*

✓ The StandardToken contract explains the proper use of this function in the comments.

Avoid invalid state by re-entering functions

> *Contracts can hand over control to another contract (e.g. by calling send or transfer). However, the second contract can again re-enter the first contract. Therefore, updates to the contracts state should be done before handing over control to another contract. Developers have to be aware of this behavior and make use of the checks-effects-interactions pattern. This means that the contracts checks its state first, afterwards the contract state is updated and any interactions with other contracts are done last.*

✓ The send() and transfer() function are not used within the contract and the relocate() function follows the checks-effects-interactions pattern.

Do not use of send function without checking return value

> *The send() function does not throw an exception if the call stack is depleted but rather returns false in that case. Therefore, the return value of send always has to be checked. We recommend using the transfer() function since it throws an exception in case of an error.*

✓ The contract does not use the send() function.

Use Safe math operations

> *Arithmetic operations do not throw an exception when an overflow occurs. Therefore, all arithmetic operations have to check for a potential overflow to avoid invalid calculations. It is recommended to use a SafeMath library that exposes safe functions for the basic operations.*

✓ The SafeMath library implements the operations for add, sub and mul. All critical math operations in the contract use the safe math equivalent.

Do not rely on block timestamps

> *The block timestamp can be influenced by the miners to some degree. Therefore, one should no rely on the block timestamp and use the block number instead.*

✓ The block timestamp is not used within the contract.

Do not use transaction origin

Checking against tx.origin for authorization provides an attack vector that can be used to trick the contract owner into unknowingly calling the contract. Therefore, one should always use msg.sender to mitigate against those kind of attacks.

✓ The transaction origin is not used within the contract.

Style

Write readable code

The overall contract should be well structured and the code should be easy to read. All variable and function names should be meaningful.

✓ The overall contract complexity is low and makes it easy to read and understand the contract. The SafeMath implementation contains only the basic operations needed by the contract.

✓ The StandardToken has been implemented according to the ERC20 Token standard and the functions check whether the pre-conditions are fulfilled before transferring tokens.

✓ The StarambraToken contract is the most complex part of the overall contract. However, the additional functionality is reasonable.

✓ With about 300 lines the contract is very readable and understandable. All functions and variables have meaningful names.

Do not write duplicate Code

Duplicate code can lead to bugs and security vulnerabilities. Therefore, it is advised to transfer duplicate code into a separate and re-usable function or modifier.

The overall contract is well structured and does not contain any complex duplicate code that can lead to diverging program logic.

● The deliverTokens() and deliverTokensBatch() functions of the Starambra contract use basically the same code to deliver tokens. The code duplication is due to improve the performance when delivering tokens to multiple users. We verified that both implementations have the same logic.

Check for Solidity compiler warnings

> *The solidity compiler does some analysis when compiling the contract to check for common errors. Any of those warnings should be avoided or at least be checked in detail to ensure that they do not point to a flaw in the implementation.*

✓ The contract does not produce any compiler warnings.

Library usage

> *Library definitions can be used to bundle functions that are independent of a contract. A good example is a SafeMath library that provides safe arithmetic operations.*

✓ The SafeMath implementation is implemented as a library definition instead of a contract. This increases code readability.

Token Sale Terms

Initial number of Token

> *The Token Sale Terms state: "The maximum number of STARAMBA.Tokens is initially restricted to 1 billion."*

✓ The initial number of tokens that can be delivered is restricted to 1,000,000,000 tokens. This number can further be increased to 1,200,000,000, see below.

Reserve for passive supply

> *The Token Sale Terms state that additional 200,000,000 tokens can be issued in chunks of 50,000,000 tokens.*

The contract allows to issue additional 50,000,000 by calling the adjustCap() function.

NOTE: The condition to generate these additional tokens is decided offline and can not be enforced within the contract.

✓ The adjustCap() function can be called four times in order to generate additional 50,000,000 tokens each time..

Maximum number of tokens (hard capped)

According to the Token Sale Terms, the maximum number if tokens is 1,200,000,000.

✓ Given the initial number of tokens (1,000,000,000) and the number of additional tokens that can be generated (200,000,000) the maximal number of tokens is capped at 1,200,000,000 tokens.

NOTE: See Finding #7 above.

Token Delivery

The contract does not allow to buy tokens directly via sending ETH to the token contract.

All tokens of the contract are delivered via the deliverTokens() and deliverTokensBatch() function. These functions can only be called by the “vendor” accounts of the contract. It is the responsibility of the vendor accounts to deliver the tokens to the end user.