StackOS

StackToken

SMART CONTRACT AUDIT

ZOKYO.

April 22th, 2021 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

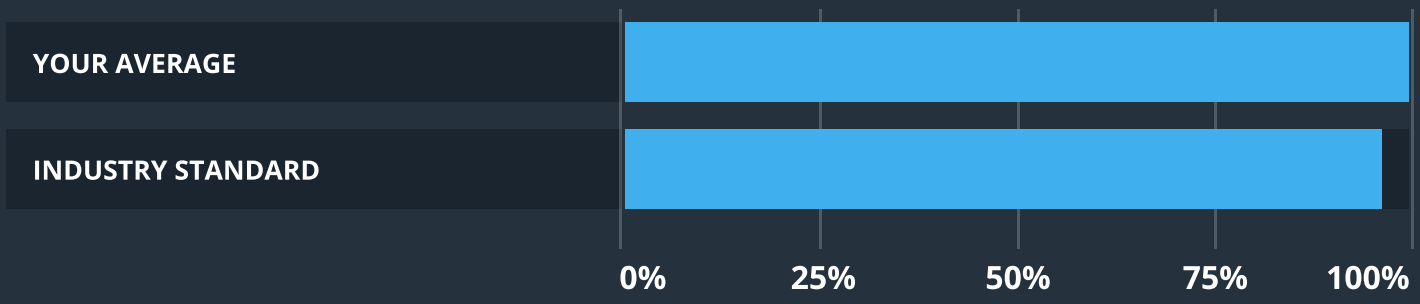
This document outlines the overall security of the StackToken smart contract, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the StackOS smart contract codebase for quality, security, and correctness.

Contract Status

There were no critical issues found during the audit.

Testable Code



The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the StackOS team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the StackOS repository.

Repository - https://gitlab.com/integro_io/idal/stackos-smart-contracts
Branch - feat/new-smart-contracts

Commit id - 1104e662ed7553f428fcd2fead49d9e2d853a086

Last commit reviewed - 32ce36257e1d48d9febbbca7daa68ab378d554e8

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of StackOS smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

There were no critical issues found during the audit. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner.

Nevertheless, contract had high-risk issues with the logic of votes delegation. The findings during the audit have a slight impact on contract performance and security.

Nevertheless, StackOS team has successfully fixed high-, medium- and low-risk issues.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

HIGH | RESOLVED

Votes delegation has no connection to transfer functionality. Double spending ability.
For now, there is questionable scenarios:
1) User1 delegates his votes to User2.
2) User1 transfers his tokens to User3.
3) User3 delegate votes for User4.
4) User2 has additional delegated votes from User1, and User4 has additional delegated votes - double-spending of votes.
Voting power saved in checkpoints should be connected to transfer functionality as well.

Recommendation:
Remove delegated votes from delegatee when delegator transfers his tokens. For example use _moveDelegates() for overloaded transfer() and transferFrom() (or for internal _transfer()) like it is done in the constructor.

MEDIUM | RESOLVED PER CONVERSATION

Delegation can be blocked

There is an ability to block delegation after the amount of tokens increasing in the wallet.
Scenario:
User1 delegates votes to User2
User1 increases token balance by achieving more StackTokens
User1 re-delegates votes to User3
_moveDelegates() reverts the transaction on Line 204, because of an attempt to subtract more tokens than delegated to User2

Recommendation:
Provide the functionality to delegate a certain amount of tokens instead of full balance and/or re-check the calculations in _moveDelegates() for sources of the votes.

Post-audit review:
After the conversation with the StackOS team we have concluded that the functionality works as intended.

LOW | RESOLVED

Misleading documentation

StackToken.sol, line 56
Function delegates() has incorrect documentation as a result of copy-paste mistake.

Recommendation:
Fix the documentation.

LOW | RESOLVED

Missing votes for self.

StackToken.sol, line 132
function getCurrentVotes() by the documentation returns the number of votes for the user. Though it returns only the number of delegated votes. For example, the user has tokens and has no delegated votes. But since he owns some tokens, he should have votes. Though such functionality (votes for token holding is missing). So, there is one of two issues:
1) There is no functionality, which calculates votes based on the amount of tokens the user owns (not only delegated) - and the issue should be considered as Medium.
2) There is misleading documentation - functionality works as planned, the user can own only the delegated votes (not based on his own tokens balance), so the description should be changed.

Recommendation:
Check if the number of votes for the user should include his own balance (in case if he hasn't delegated votes) and/or fix the documentation.

INFORMATIONAL | UNRESOLVED

Optimization in getPriorVotes()

Optimization can be provided to restrict the call for 0 address and for the block earlier of the approximate contract creation block.

Recommendation:
Provide restriction for block number and 0 address.

INFORMATIONAL | UNRESOLVED

Missing revert for delegating 0 or to zero address

For now, the contract allows performing the transaction for delegating 0 tokens or for 0 address. It creates misleading transactions with no effect.

Recommendation:
Provide require() statements to prohibit delegating 0 tokens or for 0 address to save gas and prevent misleading transactions.

INFORMATIONAL | UNRESOLVED

Missing revert for delegating votes for self.

The contract allows transactions with no effect while delegating voting power for self. It creates misleading transactions with no effect.

Recommendation:
Provide require() statements to prohibit delegating to self.

	StackToken
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

As part of our work assisting StackOS in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the StackToken contract requirements for details about issuance amounts and how the system handles these.

```
StackToken testset
  Initialization->
    √ Sets correct properties (327ms)
  Methods->
    √ Should transfer properly (972ms)
    √ Fails in case of incorrect transaction (882ms)
    √ Must delegate votes from sender to delegatee (229ms)
    √ Must delegate votes from signature to delegatee (154ms)
    √ Changes delegation correct (215ms)
    √ Changes delegation correct in the same block (578ms)
    √ Signature is assigned only for delegatee
    √ Gets current votes properly (92ms)
    √ Gets prior votes properly (1374ms)


10 passing (6s)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	100	82.14	100	100	
StackToken.sol	100	82.14	100	100	
All files	100	82.14	100	100	

We are grateful to have been given the opportunity to work with the StackOS team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the StackOS team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.