Pandora

SMART CONTRACT AUDIT

ZOKYO.

May 27th, 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.

SCORE

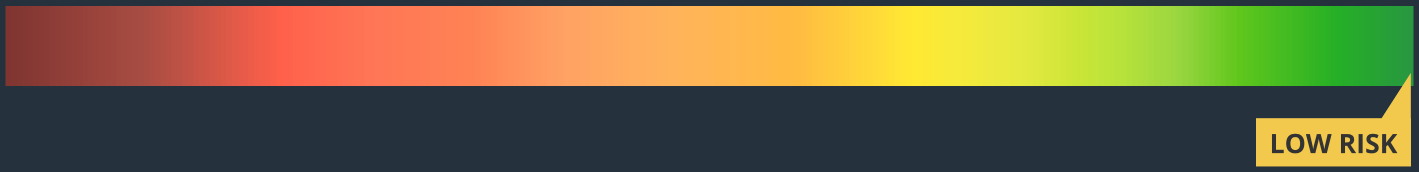
100

TECHNICAL SUMMARY

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

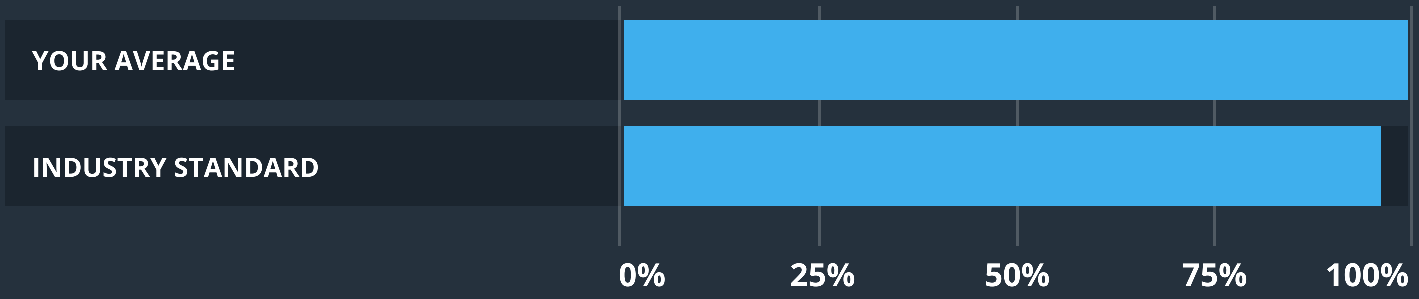
The scope of this audit was to analyze and document the Pandora smart contract codebase for quality, security, and correctness.

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 Pandora 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 Pandora repository.

Repository: https://github.com/Pandora-Finance/TokenContract

Commit audited - 3049285ac46923bc849cc30b707eb647dee362b4
Re-audit - 9ba1895587c278229e61c3fb48d60901d2195eff

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 Pandora 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.

2 Cross-comparison with other, similar smart contracts by industry leaders.

3 Testing contract logic against common and uncommon attack vectors.

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.

The findings during the audit have a slight impact on contract performance and code style.

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

Not standard ERC20

For now the protocol uses a modified version of ERC20. The contract is obtained from the OpenZeppelin library and contains its comments, though it is modified, has no key fields (which are moved to the token) and constructor. In general it is a bad practice to modify standard contracts, because the logic can be lost in further development or modified incorrectly. Good practice is to inherit the complete logic from ERC20 and provide modification in child methods.

Recommendation:
Restore standard ERC20 contract, move name, symbol and decimals fields back to the original contract and use constructor in order to set those values.

MEDIUM | RESOLVED

Solidity version update

The solidity version should be updated. Throughout the project (including interfaces). Issue is classified as Medium, because it is included to the list of standard smart contracts' vulnerabilities. Currently used version (0.5.17) is considered obsolete, which contradicts the standard checklist.

Recommendation:
You need to update the solidity version to the latest one in the branch - consider 0.6.12 or 0.7.6 or 0.8.4

LOW | RESOLVED

Move decimals to the variable

Move the expression “10 ** uint256(decimals)” to the totalTokensAmount variable in order to minimize in-code calculations and increase the code style. Also, since decimals are set just once and there is no interface to change it, the number can be used instead of decimals variable.

Recommendation:
Move decimals multiplier to the variable and use number instead of the variable.

LOW | RESOLVED

Use constant value

totalTokensAmount can be set as constant. It will perform gas optimization and increase the code style.

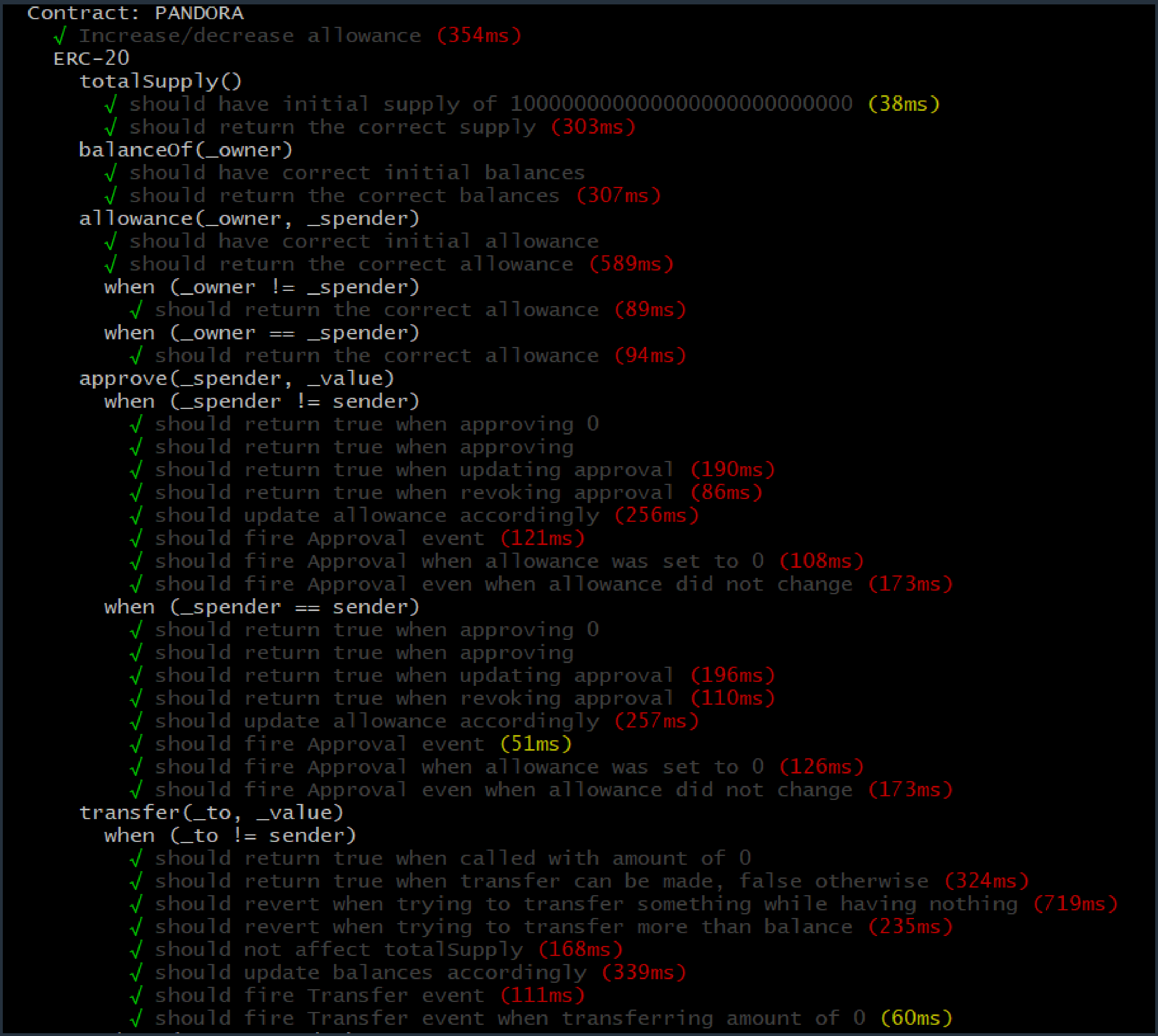
Recommendation:
Set totalTokensAmount as constant.

	PandoraToken
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

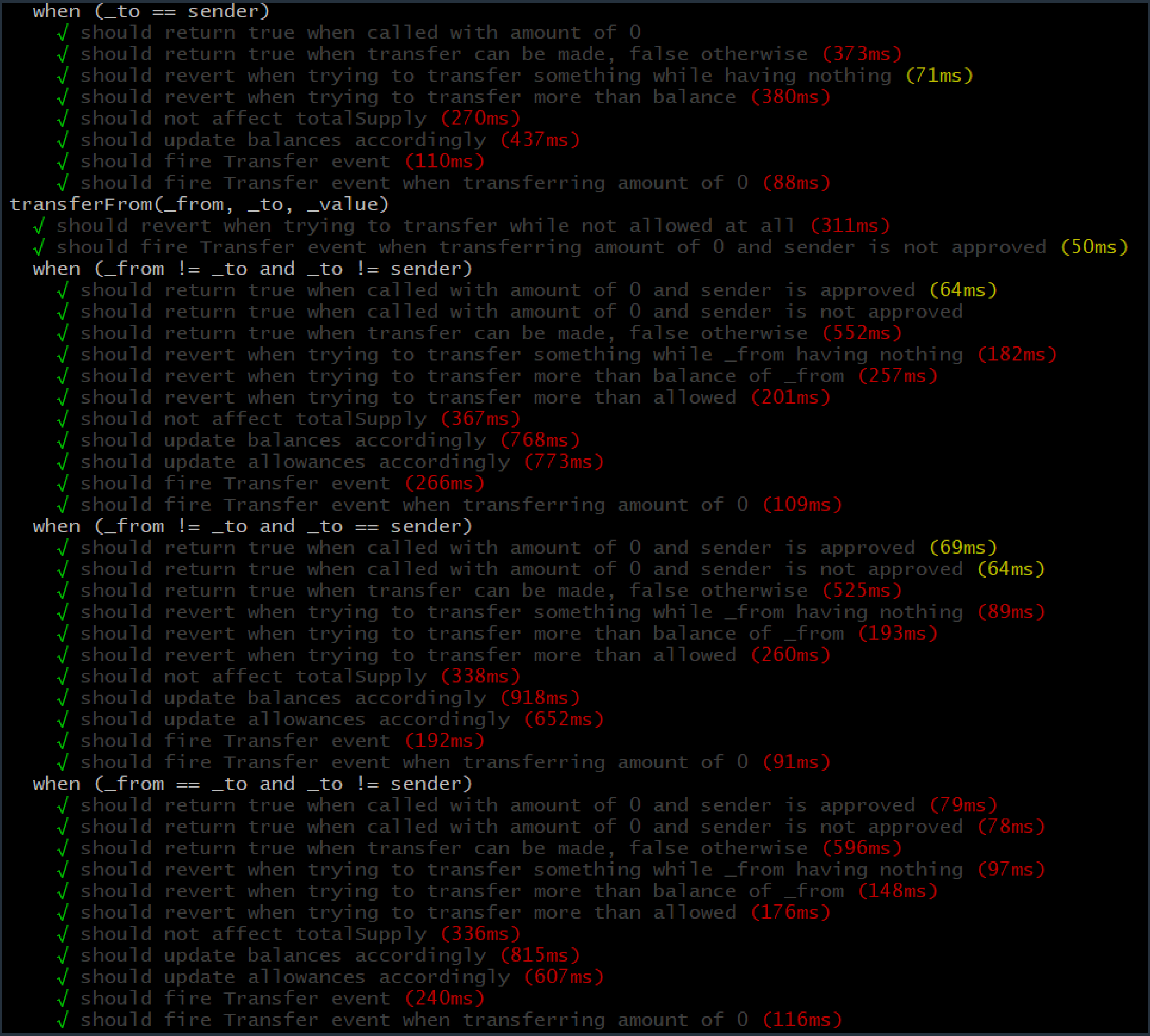
CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo team

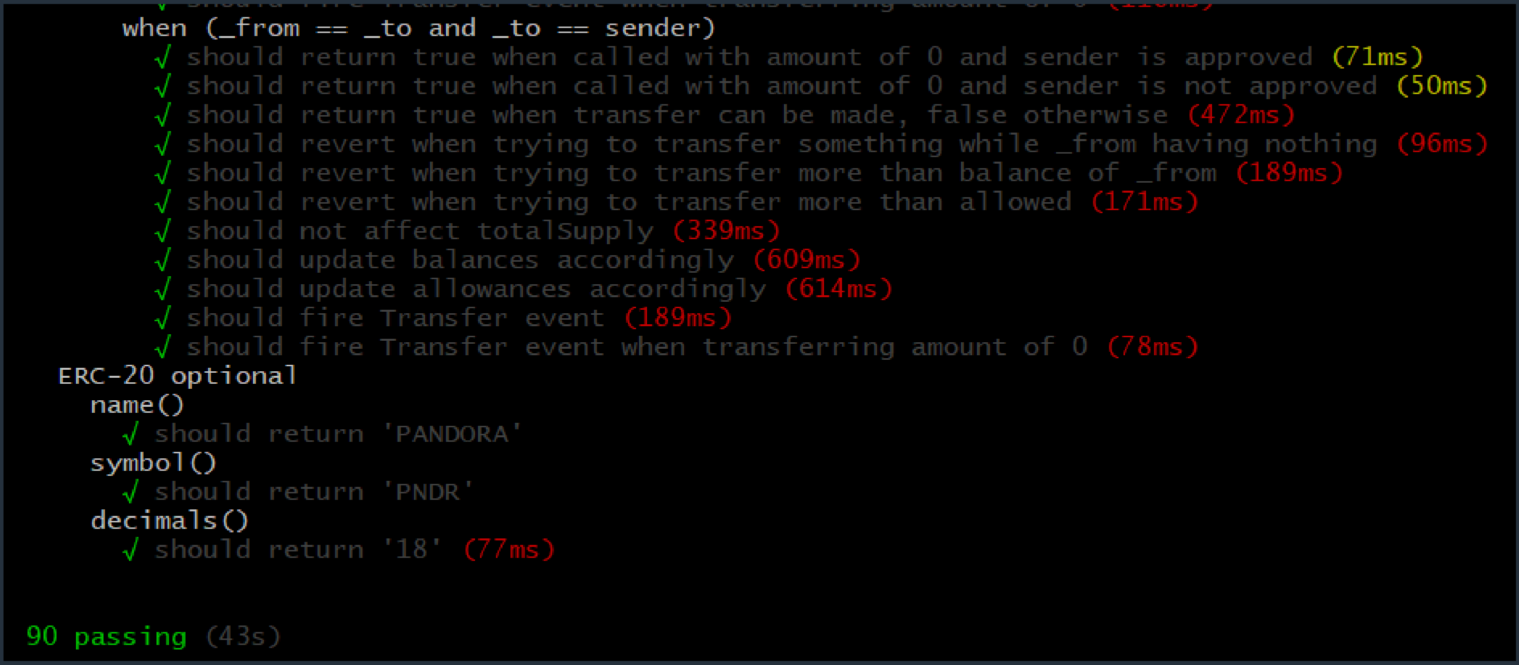
As part of our work assisting Pandora in verifying the correctness of their contract code, our team was responsible for writing integration tests using Truffle testing framework.
Tests were based on the functionality of the code, as well as review of the Pandora contract requirements for details about issuance amounts and how the system handles these.



```
Contract: PANDORA
  √ Increase/decrease allowance (354ms)
  ERC-20
    totalSupply()
      √ should have initial supply of 1000000000000000000000000000 (38ms)
      √ should return the correct supply (303ms)
    balanceOf(_owner)
      √ should have correct initial balances
      √ should return the correct balances (307ms)
    allowance(_owner, _spender)
      √ should have correct initial allowance
      √ should return the correct allowance (589ms)
      when (_owner != _spender)
        √ should return the correct allowance (89ms)
      when (_owner == _spender)
        √ should return the correct allowance (94ms)
    approve(_spender, _value)
      when (_spender != sender)
        √ should return true when approving 0
        √ should return true when approving
        √ should return true when updating approval (190ms)
        √ should return true when revoking approval (86ms)
        √ should update allowance accordingly (256ms)
        √ should fire Approval event (121ms)
        √ should fire Approval when allowance was set to 0 (108ms)
        √ should fire Approval even when allowance did not change (173ms)
      when (_spender == sender)
        √ should return true when approving 0
        √ should return true when approving
        √ should return true when updating approval (196ms)
        √ should return true when revoking approval (110ms)
        √ should update allowance accordingly (257ms)
        √ should fire Approval event (51ms)
        √ should fire Approval when allowance was set to 0 (126ms)
        √ should fire Approval even when allowance did not change (173ms)
    transfer(_to, _value)
      when (_to != sender)
        √ should return true when called with amount of 0
        √ should return true when transfer can be made, false otherwise (324ms)
        √ should revert when trying to transfer something while having nothing (719ms)
        √ should revert when trying to transfer more than balance (235ms)
        √ should not affect totalSupply (168ms)
        √ should update balances accordingly (339ms)
        √ should fire Transfer event (111ms)
        √ should fire Transfer event when transferring amount of 0 (60ms)
```

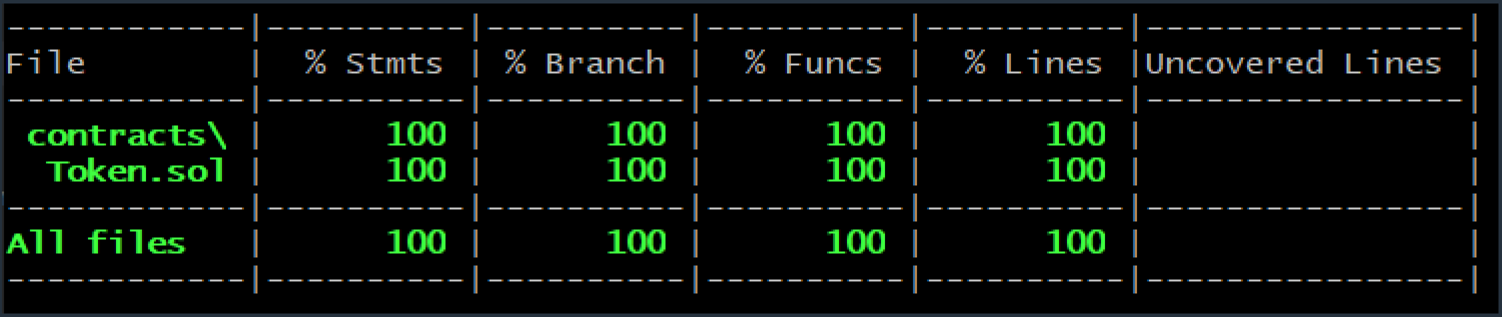


```
  when (_to == sender)
    √ should return true when called with amount of 0
    √ should return true when transfer can be made, false otherwise (373ms)
    √ should revert when trying to transfer something while having nothing (71ms)
    √ should revert when trying to transfer more than balance (380ms)
    √ should not affect totalSupply (270ms)
    √ should update balances accordingly (437ms)
    √ should fire Transfer event (110ms)
    √ should fire Transfer event when transferring amount of 0 (88ms)
transferFrom(_from, _to, _value)
  √ should revert when trying to transfer while not allowed at all (311ms)
  √ should fire Transfer event when transferring amount of 0 and sender is not approved (50ms)
  when (_from != _to and _to != sender)
    √ should return true when called with amount of 0 and sender is approved (64ms)
    √ should return true when called with amount of 0 and sender is not approved
    √ should return true when transfer can be made, false otherwise (552ms)
    √ should revert when trying to transfer something while _from having nothing (182ms)
    √ should revert when trying to transfer more than balance of _from (257ms)
    √ should revert when trying to transfer more than allowed (201ms)
    √ should not affect totalSupply (367ms)
    √ should update balances accordingly (768ms)
    √ should update allowances accordingly (773ms)
    √ should fire Transfer event (266ms)
    √ should fire Transfer event when transferring amount of 0 (109ms)
  when (_from != _to and _to == sender)
    √ should return true when called with amount of 0 and sender is approved (69ms)
    √ should return true when called with amount of 0 and sender is not approved (64ms)
    √ should return true when transfer can be made, false otherwise (525ms)
    √ should revert when trying to transfer something while _from having nothing (89ms)
    √ should revert when trying to transfer more than balance of _from (193ms)
    √ should revert when trying to transfer more than allowed (260ms)
    √ should not affect totalSupply (338ms)
    √ should update balances accordingly (918ms)
    √ should update allowances accordingly (652ms)
    √ should fire Transfer event (192ms)
    √ should fire Transfer event when transferring amount of 0 (91ms)
  when (_from == _to and _to != sender)
    √ should return true when called with amount of 0 and sender is approved (79ms)
    √ should return true when called with amount of 0 and sender is not approved (78ms)
    √ should return true when transfer can be made, false otherwise (596ms)
    √ should revert when trying to transfer something while _from having nothing (97ms)
    √ should revert when trying to transfer more than balance of _from (148ms)
    √ should revert when trying to transfer more than allowed (176ms)
    √ should not affect totalSupply (336ms)
    √ should update balances accordingly (815ms)
    √ should update allowances accordingly (607ms)
    √ should fire Transfer event (240ms)
    √ should fire Transfer event when transferring amount of 0 (116ms)
```



```
          √ should fire Transfer event when transferring amount of 0
        when (_from == _to and _to == sender)
          √ should return true when called with amount of 0 and sender is approved (71ms)
          √ should return true when called with amount of 0 and sender is not approved (50ms)
          √ should return true when transfer can be made, false otherwise (472ms)
          √ should revert when trying to transfer something while _from having nothing (96ms)
          √ should revert when trying to transfer more than balance of _from (189ms)
          √ should revert when trying to transfer more than allowed (171ms)
          √ should not affect totalSupply (339ms)
          √ should update balances accordingly (609ms)
          √ should update allowances accordingly (614ms)
          √ should fire Transfer event (189ms)
          √ should fire Transfer event when transferring amount of 0 (78ms)
    ERC-20 optional
      name()
        √ should return 'PANDORA'
      symbol()
        √ should return 'PNDR'
      decimals()
        √ should return '18' (77ms)


  90 passing (43s)
```



```
------------|----------|----------|----------|----------|----------------|
File        |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
------------|----------|----------|----------|----------|----------------|
 contracts\ |      100 |      100 |      100 |      100 |                |
  Token.sol |      100 |      100 |      100 |      100 |                |
------------|----------|----------|----------|----------|----------------|
All files   |      100 |      100 |      100 |      100 |                |
------------|----------|----------|----------|----------|----------------|
```

We are grateful to have been given the opportunity to work with the Pandora 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 Pandora team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.