TALENTA

One-stop end-to-end blockchain consulting and service customized for your project

Smart Contract Audit Report

Executive Summary

We, Talenta Pte Ltd, have audited the smart contract of Õpet Foundation. The smart contract audited passes all audit cases detailed in this report and we have not found any material issues during the execution of those cases.

This smart contract audit will cover the following topics:

1. Overview
2. Code audit
3. Automation Testing
4. Vulnerability Testing
5. Attack Testing
6. Optimization
7. Summary

Overview

This document is a report of **Õpet** Foundation's smart contract audit. Testing & audit cases were run based on the forked codebase's version. This means that the audit results are only representative of the forked codebase. This report is unsuitable for any code bases dated before or after the fork.

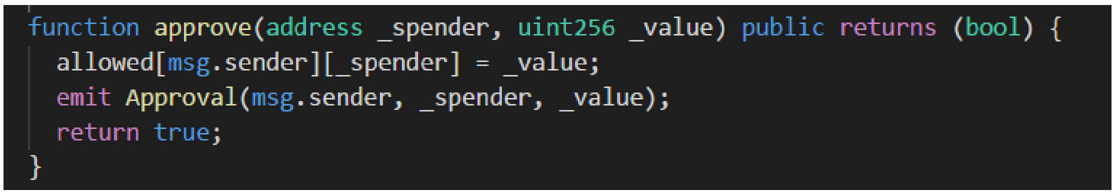
Code Audit

This section is for static solidity code audit. Contents below detail the defects we've found in the contract.

Audit Findings:

1. **APPROVE METHOD RACE CONDITIONS RESISTANCE**

   As the contract provides increaseApproval and decreaseApproval methods to avoid race conditions, the approve method should only work under allowed [msg.sender][_spender] = 0 condition. We recommend that the constraint be added.



```
function approve(address _spender, uint256 _value) public returns (bool) {
  allowed[msg.sender][_spender] = _value;
  emit Approval(msg.sender, _spender, _value);
  return true;
}
```

2. **No FALLBACK FUNCTION**

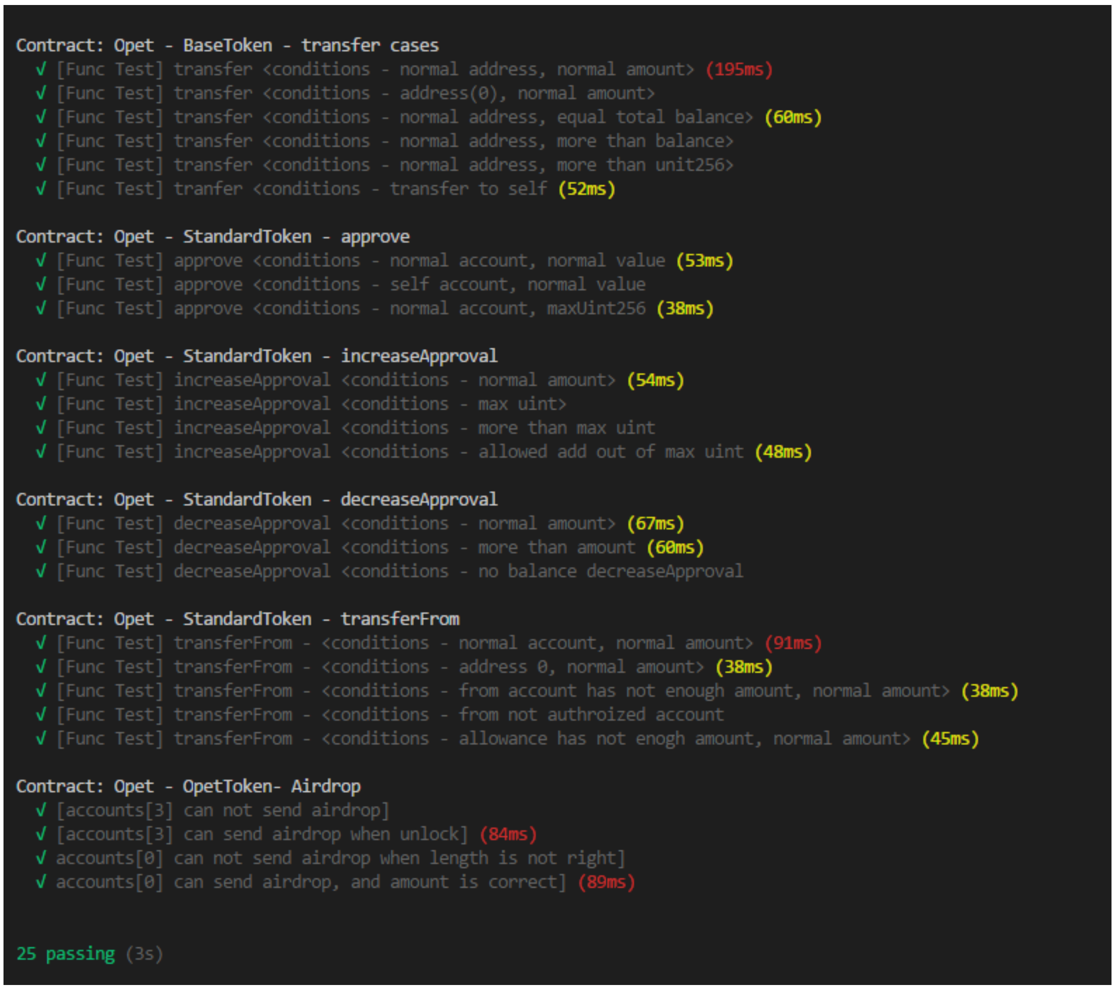
   The contract has no payable function. Any ether sent to the contract cannot be withdrawn unless the function is implemented.

Automation Audit

This section contains the outputs from applying automated test cases to the code base. It covers all methods, including positive, negative and leak testing cases for functions. The automation code is attached in the appendix of this document.

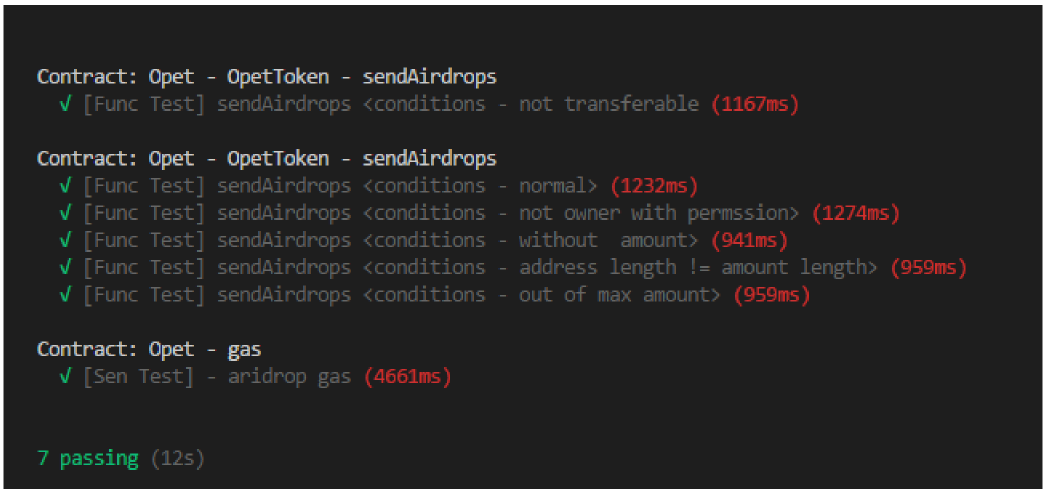
Automation Results:

1. ERC20 FUNCTIONS

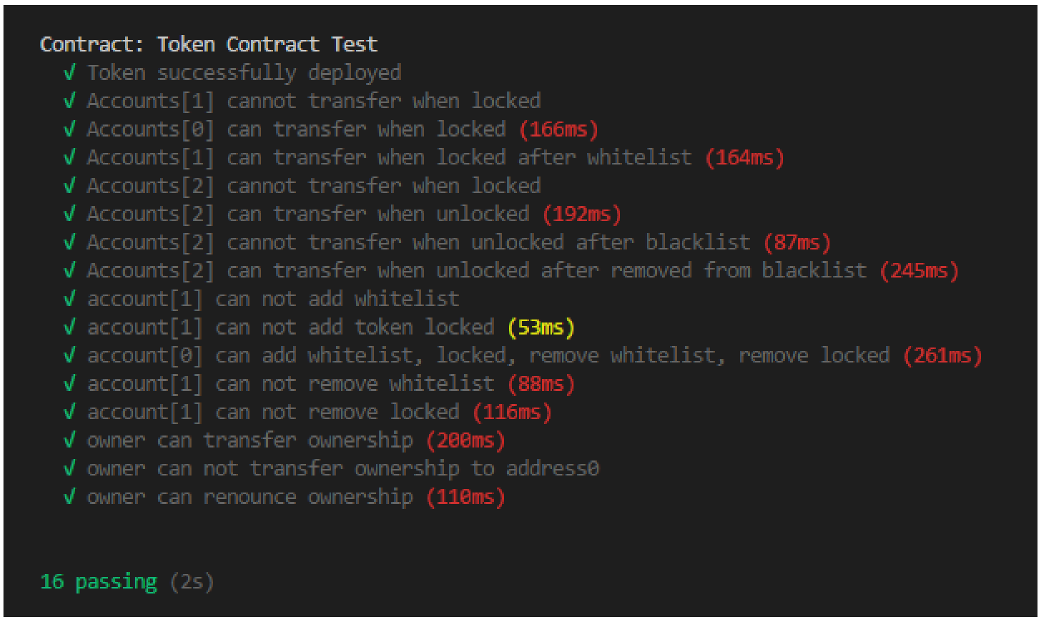


```
Contract: Opet - BaseToken - transfer cases
  √ [Func Test] transfer <conditions - normal address, normal amount> (195ms)
  √ [Func Test] transfer <conditions - address(0), normal amount>
  √ [Func Test] transfer <conditions - normal address, equal total balance> (60ms)
  √ [Func Test] transfer <conditions - normal address, more than balance>
  √ [Func Test] transfer <conditions - normal address, more than unit256>
  √ [Func Test] tranfer <conditions - transfer to self (52ms)

Contract: Opet - StandardToken - approve
  √ [Func Test] approve <conditions - normal account, normal value (53ms)
  √ [Func Test] approve <conditions - self account, normal value
  √ [Func Test] approve <conditions - normal account, maxUint256 (38ms)

Contract: Opet - StandardToken - increaseApproval
  √ [Func Test] increaseApproval <conditions - normal amount> (54ms)
  √ [Func Test] increaseApproval <conditions - max uint>
  √ [Func Test] increaseApproval <conditions - more than max uint
  √ [Func Test] increaseApproval <conditions - allowed add out of max uint (48ms)

Contract: Opet - StandardToken - decreaseApproval
  √ [Func Test] decreaseApproval <conditions - normal amount> (67ms)
  √ [Func Test] decreaseApproval <conditions - more than amount (60ms)
  √ [Func Test] decreaseApproval <conditions - no balance decreaseApproval

Contract: Opet - StandardToken - transferFrom
  √ [Func Test] transferFrom - <conditions - normal account, normal amount> (91ms)
  √ [Func Test] transferFrom - <conditions - address 0, normal amount> (38ms)
  √ [Func Test] transferFrom - <conditions - from account has not enough amount, normal amount> (38ms)
  √ [Func Test] transferFrom - <conditions - from not authroized account
  √ [Func Test] transferFrom - <conditions - allowance has not enogh amount, normal amount> (45ms)

Contract: Opet - OpetToken- Airdrop
  √ [accounts[3] can not send airdrop]
  √ [accounts[3] can send airdrop when unlock] (84ms)
  √ accounts[0] can not send airdrop when length is not right]
  √ accounts[0] can send airdrop, and amount is correct] (89ms)


25 passing (3s)
```

2. CUSTOM FUNCTIONS



```
Contract: Opet - OpetToken - sendAirdrops
  √ [Func Test] sendAirdrops <conditions - not transferable (1167ms)

Contract: Opet - OpetToken - sendAirdrops
  √ [Func Test] sendAirdrops <conditions - normal> (1232ms)
  √ [Func Test] sendAirdrops <conditions - not owner with permssion> (1274ms)
  √ [Func Test] sendAirdrops <conditions - without  amount> (941ms)
  √ [Func Test] sendAirdrops <conditions - address length != amount length> (959ms)
  √ [Func Test] sendAirdrops <conditions - out of max amount> (959ms)

Contract: Opet - gas
  √ [Sen Test] - aridrop gas (4661ms)


7 passing (12s)
```



```
Contract: Token Contract Test
  √ Token successfully deployed
  √ Accounts[1] cannot transfer when locked
  √ Accounts[0] can transfer when locked (166ms)
  √ Accounts[1] can transfer when locked after whitelist (164ms)
  √ Accounts[2] cannot transfer when locked
  √ Accounts[2] can transfer when unlocked (192ms)
  √ Accounts[2] cannot transfer when unlocked after blacklist (87ms)
  √ Accounts[2] can transfer when unlocked after removed from blacklist (245ms)
  √ account[1] can not add whitelist
  √ account[1] can not add token locked (53ms)
  √ account[0] can add whitelist, locked, remove whitelist, remove locked (261ms)
  √ account[1] can not remove whitelist (88ms)
  √ account[1] can not remove locked (116ms)
  √ owner can transfer ownership (200ms)
  √ owner can not transfer ownership to address0
  √ owner can renounce ownership (110ms)


16 passing (2s)
```

Vulnerability Testing

This section details the results of vulnerability testing done on the smart contract, including integer overflow and custom logic testing.

Integer Overflow Testing Cases

Case Name	Result
approve	Pass
increaseApprove	Pass
descreaseApprove	Pass

transfer	Pass
transferFrom	Pass
sendAirdrops	Pass

Attack Testing

This section details the outcome of security testing that simulates attacks on the smart contract. The attack points and methods are listed in this section. We cannot guarantee that the smart contract is safe under attack points and methods not listed here.

1. **REENTRANCY**
   No reentrancy issues have been found in the contract. The transfer and transferFrom functions do not call methods to send tokens.
2. **SHORT ADDRESS ATTACK**
   No short address attack code has been found in the contract. For details on short address attacks, refer to https://github.com/OpenZeppelin/openzeppelin-solidity/pull/188.
   Notice: Solidity 0.5.0 fixed this issue in the language. However, the contract is uses Solidity 0.4.23. The choice of adding protection code or leaving it out to save on gas depends on the client. This is a low-risk issue.

Optimization

This section contains our suggestions for this smart contract. These proposals are not defects but can be valuable in improving this smart contract.

Gas Use Optimization

1. **SEVERAL FUNTIONS WILL COST OUT OF GAS LIMITATION**
   - sendAirdrops function loops transfer which costs N times gas for successful sending.
   - addWhitelistedTransfer/removeWhitelistedTransfer/addToTokenLocked/removeFromTokenLocked functions loop updating of variables which cost N times gas for successful updating.

   In our testing, the maximum total address count is 200 addresses before it reaches the gas limit per block. Any additional addresses will cause out of gas failure.

Business Logic Optimization

1. **SEVERAL FUNCTIONS WITH NO RETURN VALUE**
   sendAirdrops/addWhitelistedTransfer/removeWhitelistedTransfer/addToTokenLocked/removeFromTokenLocked methods do not return a value. It would be better to return a Boolean value to specify whether the methods were called successfully.
2. **NOPAUSED APPROVE/INCREASE APPROVAL/DECREASE APPROVAL**
   Transfer and transferFrom methods have transferable modifiers, but related functions approve, increaseApprove and descreaseApprove don’t have transferable modifiers. As they’re all ECR20 base token methods, they should be treated using the same restriction.
3. **NO PAUSE METHOD**
   The contract has unpauseTransfer method but no pausedTransfer method. It would be better to implement it.

Summary

The smart contract passes all audit cases above and we have not found any material issues during the execution of those cases.