Shapeshift

SMART CONTRACT AUDIT

ZOKYO.

June 23d, 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

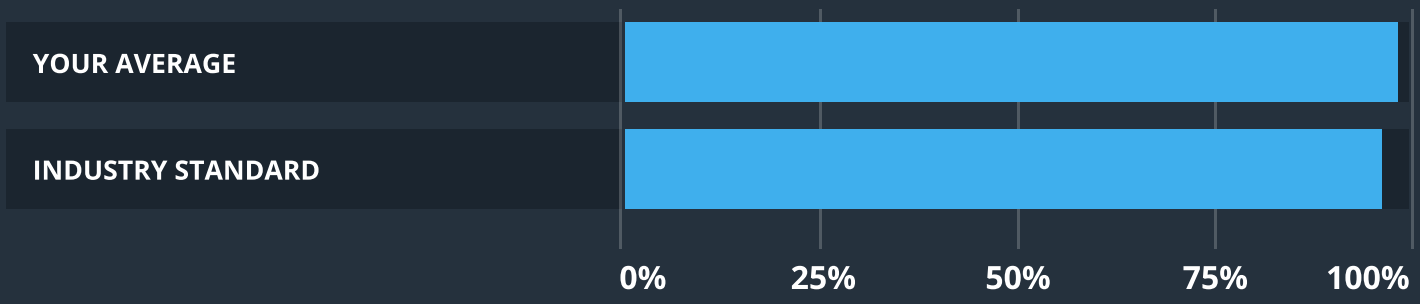
This document outlines the overall security of the Shapeshift airdrop smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Shapeshift airdrop 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 97%, 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 Shapeshift 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 Airdrop smartt contract's source code was taken from two archives provided by the Shapeshift's team (initial archive and updated archive after fixes). Archives' hashes are given below:
SHA-256 (initial): 8e9f897ebf9217382caecab54f23b9d304ba6372b2263f0f0129ff7f6666d5d5
SHA-256(post-audit):
227946d6dd51924d216a6c8a09bb5dbf036a5f07b16512097c1ca0d55aea5421
Within the scope of this audit Zokyo auditors have reviewed the following contract(s):
MerkleDistributor.sol
TokenDistributor.sol

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 Shapeshift 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. Nevertheless, there were found several high and medium risk issues connected to missed checks, commented code and unclear functionality. Also, contracts set required Solidity version upgrade. All tother he mentioned findings may have an effect only in case of specific conditions performed by the contract owner or are connected to the code style, extra variables and minor optimizations.

Nevertheless, all high risk findings were successfully fixed by the Shapeshift team.
Though several minor issues were left unresolved.

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

Commented code in production

TokenDistribution.sol, line 122.
There is a commented check which can influence a process of claiming. Rather the account itself can perform a claim (so this check should be uncommented), or anyone can perform a claim on behalf of another account (just with correct merkle proof).

Recommendation:
Review commented “require” statement and confirm, that it should be removed or included to the code.

Post-audit
Functionality verified by the customer.

MEDIUM | RESOLVED

Solidity version update

Issue is classified as Medium, because it is included to the list of standard smart contracts’ vulnerabilities. Currently used version (0.6.1) is not the last in the line, so it contradicts standard checklist.

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

MEDIUM | UNRESOLVED

No security checks for the final epoch calculation

TokenDistribution.sol, line 51.
With incorrectly handled parameters there is a possibility to get underflow during the epoch calculation, since no SafeMath applied in this place nor necessary checks performed.

Recommendation:
Add necessary “require” statements to mitigate the case when “currentRewardRate / rewardReductionPerEpoch_” equals 0 or add SafeMath.

LOW | UNRESOLVED

Methods should be declared as external

In order to decrease the gas usage, next methods should be declared as external:
TokenDistributor.initialize() (TokenDistributor.sol#27-52)
TokenDistributor.getClaimsStartTime() (TokenDistributor.sol#61-63)
TokenDistributor.getNextEpochStart() (TokenDistributor.sol#66-74)
TokenDistributor.getTimeUntilNextEpoch() (TokenDistributor.sol#76-84)
TokenDistributor.getNextEpochRewardsRate() (TokenDistributor.sol#106-111)

Recommendation:
Declare methods as external.

LOW | RESOLVED

Unnecessary Migration

Migration.sol contract can be removed to increase the readability of the code and in order to avoid errors during migrations.

Recommendation:
Remove Migration.sol.

LOW | UNRESOLVED

Extra variable

TokenDistributor.sol, line 49: variable currentRewardRate can be completely omitted. It is initialized just once, it is used just once and it is never changed.

Recommendation:
Remove currentRewardRate or set as constant.

INFORMATIONAL | UNRESOLVED

Condition never met

TokenDistributor.sol, line 108, getNextEpochRewardsRate()
if (epoch == 0) return MAX_BPS;
The condition is never met. epoch is always set to at least 1.

Recommendation:
Check the value of the epoch and the condition and/or remove the condition.

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo team

As part of our work assisting Shapeshift 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 Shapeshift contract requirements for details about issuance amounts and how the system handles these.

```
TokenDistributor test
  Should be executed without additional errors.
    √ Should return correct values. (1274ms)
    √ Should call functions without error. (798ms)
    √ Should call functions with additional "if" and raise an error if claim period not finished. (1484ms)
    √ Should call functions without error. (1058ms)
    √ Should raise an error in cases of past rewards claim period. (738ms)
    √ Should raise an error in cases of before claim start. (839ms)
    √ Should allow a claim, or raise an error if a claim has already occurred. (1271ms)
    √ Should raise an error in cases of transfer to user failed. (1018ms)
    √ Should raise an error in cases of invalid proof. (727ms)

MerkleDistributor test
  Should be executed without additional errors.
    √ Should allow a claim, or raise an error if a claim has already occurred. (1054ms)
    √ Should raise an error in cases of invalid proof. (698ms)
    √ Should raise an error in cases of transfer failed. (671ms)


12 passing (13s)
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	98.51	94.12	100	100	
MerkleDistributor.sol	100	100	100	100	
TokenDistributor.sol	98.04	92.86	100	100	
All files	98.51	94.12	100	100	

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

ZOKYO.