SMART CONTRACT AUDIT

ZOKYO.

October 27th 2022 | v. 1.0

PASS

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

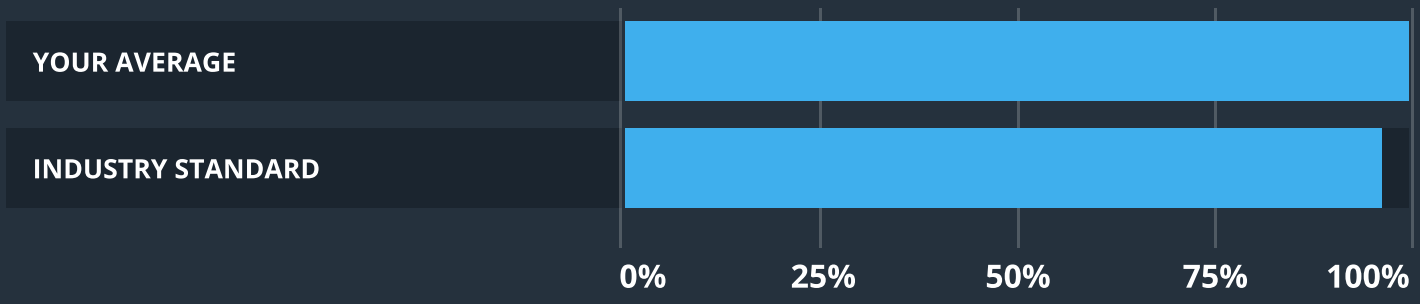
TECHNICAL SUMMARY

This document outlines the overall security of the TrustSwap smart contracts evaluated by the Zokyo Security team.

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

Contract Status

Testable Code



The 100% of the code is testable, which corresponds the 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 recommend that the TrustSwap team put in place a bug bounty program to encourage further active analysis of the smart contract.

TABLE OF CONTENTS

Auditing Strategy and Techniques Applied 3
Executive Summary 5
Protocol Overview 6
Structure and Organization of the Document 13
Complete Analysis 14
Code Coverage and Test Results for all files (by the Zokyo Security team) 17
Code Coverage and Test Results for all files (by the TrustSwap) 19

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the Vesting smart contracts was taken from the TrustSwap repository.
https://github.com/trustswap/vesting-contracts/tree/updated-vesting

Initial commit: 2e84a90e9c45d389acd0606a08b205f580617ab1
Final commit: cc15c4665623cf614453c44886b79b86280053e7

Within the scope of this audit, Zokyo auditors have reviewed the following contract(s):

- src\Vesting.sol
- src\VestingFactory.sol

Throughout the review process, Zokyo Security ensures that the contract:

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

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of TrustSwap smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented any issues as they were discovered. A part of this work included 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

Zokyo Security has checked the set of contracts from TrustSwap. The contracts represent a standard linear vesting logic with Merkle-proofs-based distribution and additional functionality for vesting revoke. Vesting is fully autonomous, though the vesting owner can stop vesting for specific users if they are not marked as unrevocable. The contracts also contain a factory for vesting contracts production. It allows the user to create vesting based on the chosen token. TrustSwap gets the fee in ETH upon vesting creation, and after that, vesting becomes autonomous.

Auditors have checked the correctness of the Merkle proofs verification integration into the vesting, the correctness of the checks performed in the contract, the correctness of the fee calculation, the responsibilities of the protocol and the vesting owner. All checks were supported with appropriate tests prepared by the team of auditors.

The team has detected several issues connected with the style and quality of the code and a standard issue with ETH transfer. The TrustSwap team has fixed the security-related issue and clarified others. The security of the contracts is evaluated as High.

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 it can lead to a significant loss, funds may be lost or allocated incorrectly.

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-1 | RESOLVED

Deprecated ETH transfer.

VestingFactory.sol. handleFees()
Due to the Istanbul update, there were several changes in the EVM that made .transfer() and .send() methods deprecated for the ETH transfer.
It is highly recommended to use the .call() functionality with mandatory result check, or the built-in functionality of the Address contract from OpenZeppelin library.

Recommendation:
Always use call() to send ETH (with proper result check) or use OpenZeppelin Address library.

INFO-1 | RESOLVED

The order of conditions for gas consumption.

Vesting.sol. claim()
It is recommended to check the revoked status right after the proof of verification and before the claimable amount calculation due to the high importance of the check and on its inevitable effect.
Vesting.sol, stopVesting()
The same applies for the check if the user is un-revocable. It is recommended to check the “revocable” variable immediately after the proof of verification and before the “getRevoked()” check.

Recommendation:
Consider changing the order of conditions.

INFO-2 | ACKNOWLEDGED

Use constants.

VestingFactory.sol. handleFees(), line 66
Use constants instead of plain numbers (100, 5) to increase the readability and quality of the code.

Recommendation:
Consider using constants.

Post-audit:
The team stated that the additional comment for that particular line with calculations has necessary information for the readability. Since the issue is connected with the code style and not security, it is marked as “acknowledged”.

INFO-3 | VERIFIED

Fee conversion logic.

VestingFactory.sol. getFeeInETH(), setFeeParams()
As per naming, feesInUSD represents the USD amount, and getFeeInETH() performs its conversion to ETH through the Uniswap oracle. However, there is no information on which pair will be used for conversion and what stablecoin as a USD analog will be used. Besides, there is no validation if the pair (uniV2Pair) actually contains a stablecoin.

Recommendation:
Verify the correctness of the conversion logic. Verify if the correct pair with a stablecoin will be used and which stablecoin will be used as a USD analog.

Post-audit:
The trustswap team verified that the owner of the contract is fully responsible for the correct pair setup (USDT/ETH).

	src\Vesting.sol	src\VestingFactory.sol
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
Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying token)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

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

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

```
Vesting [percentageOnStart = 100%]
  claim()
    ✓ Users can claim reward  (107ms)
    ✓ Users can claim reward after end of vesting (86ms)
    ✓ Users cannot claim reward before start of vesting (57ms)
    ✓ Users cannot claim reward with invalid merkle proof
    ✓ Users cannot claim reward when reward is claimed already (81ms)
    ✓ Users cannot claim more reward than claimable (50ms)
    ✓ Users cannot claim reward if he revoked (179ms)
  stopVesting()
    ✓ Stop vesting for user transfer all claimable to user (49ms)
    ✓ Only owner can stop vesting
    ✓ Stop vesting for user transfer all claimable to user [Claimable  == 0] (73ms)
    ✓ Vesting cannot be stopped with invalid merkle proof
    ✓ Vesting cannot be stopped twice for the same user (57ms)
    ✓ Cannot stop ended vesting
    ✓ Cannot stop irrevocable vesting
  transferOwnership()
    ✓ Transfer Ownership
  constructor()
    ✓ Impossible to create vesting wit 0 merkle root
Vesting [percentageOnStart = 0]
  claim()
    ✓ Users can claim half reward after half time has passed  (78ms)
    ✓ Users cannot claim more reward that allowed at current moment
Vesting Factory
    ✓ Vesting Factory
    ✓ Create vesting with feesles token (72ms)
    ✓ Create vesting without feesles token (72ms)
```

✓ Create vesting send rest from fee to sender (85ms)
✓ Create vesting with fees 5% slippage (98ms)
✓ Impossible to create vesting when fee is not met (72ms)
✓ Impossible to create vesting with zero amount
✓ Company wallet cannot be set as zero
✓ only contact can be set as uniswap contract

27 passing (2s)

FILE	% STMTS	% BRANCH	% FUNCS	% LINES
Vesting.sol	100	100	100	100
VestingFactory.sol	100	100	100	100
All files	**100**	**100**	**100**	**100**

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the TrustSwap team

As a part of our work assisting the TrustSwap team in verifying the correctness of their contract code, we have checked the full set of unit tests prepared by the TrustSwap team.

It needs to be mentioned that the original code has a significant original coverage with testing scenarios provided by the TrustSwap team. All of them were also carefully checked by the team of auditors.

```
Running 19 tests for test/VestingFactory.t.sol:VestingFactoryTest
[PASS] testCreateVestingFee() (gas: 1029359)
[PASS] testCreateVestingFeeAcceptableBellow() (gas: 1028901)
[PASS] testCreateVestingFeeRefund() (gas: 1035531)
[PASS] testCreateVestingFeeless() (gas: 896566)
[PASS] testFailCreateVestingAllowance() (gas: 854506)
[PASS] testFailCreateVestingBalance() (gas: 878274)
[PASS] testFailCreateVestingFeeUnacceptableBellow() (gas: 153541)
[PASS] testFailCreateVestingZeroAmount() (gas: 14932)
[PASS] testFailCreateVestingZeroMerkleRoot() (gas: 50663)
[PASS] testFailFeelessTokenOwner() (gas: 12946)
[PASS] testFailSetFeesCompanyWalletZero() (gas: 17689)
[PASS] testFailSetFeesNoContractPair() (gas: 17652)
[PASS] testFailSetFeesNoContractRouter() (gas: 15097)
[PASS] testFeelessToken() (gas: 35345)
[PASS] testGetFee() (gas: 104657)
[PASS] testGetFeeWethFirst() (gas: 11752190)
[PASS] testNoFee() (gas: 9900)
[PASS] testSetFees() (gas: 103821)
[PASS] testUniDeployment() (gas: 19181)
Test result: ok. 19 passed; 0 failed; finished in 8.93ms

Running 20 tests for test/Vesting.t.sol:VestingTest
[PASS] testClaim() (gas: 103803)
[PASS] testClaimForOther() (gas: 109178)
[PASS] testClaimGtClaimable() (gas: 49752)
[PASS] testClaimMultiple() (gas: 370336)
[PASS] testClaimPartially() (gas: 136556)
```

```
[PASS] testClaimRevoked() (gas: 119286)
[PASS] testClaimTwoTimes() (gas: 117655)
[PASS] testClaimZero() (gas: 38167)
[PASS] testClaimZero2() (gas: 105350)
[PASS] testEmptyMerkleRoot() (gas: 38822)
[PASS] testGetClaimable(uint256,uint256,uint256,uint256,uint256,uint256)
(runs: 256, μ: 28381, ~: 28562)
[PASS] testInvalidProof() (gas: 37818)
[PASS] testStopVestingAfter() (gas: 39008)
[PASS] testStopVestingBefore() (gas: 91326)
[PASS] testStopVestingDuring(uint256) (runs: 256, μ: 116974, ~: 127842)
[PASS] testStopVestingOnlyOwner() (gas: 37961)
[PASS] testStopVestingTwice() (gas: 90678)
[PASS] testStopVestingUnrevocable() (gas: 36795)
[PASS] testTransferOwnership() (gas: 18916)
[PASS] testTransferOwnershipZeroAddress() (gas: 10673)
Test result: ok. 20 passed; 0 failed; finished in 39.12ms
```

We are grateful for the opportunity to work with the TrustSwap team.

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

Zokyo Security recommends the TrustSwap team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.