FOX Yieldy

SMART CONTRACT AUDIT

ZOKYO.

March 28th, 2022 | 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

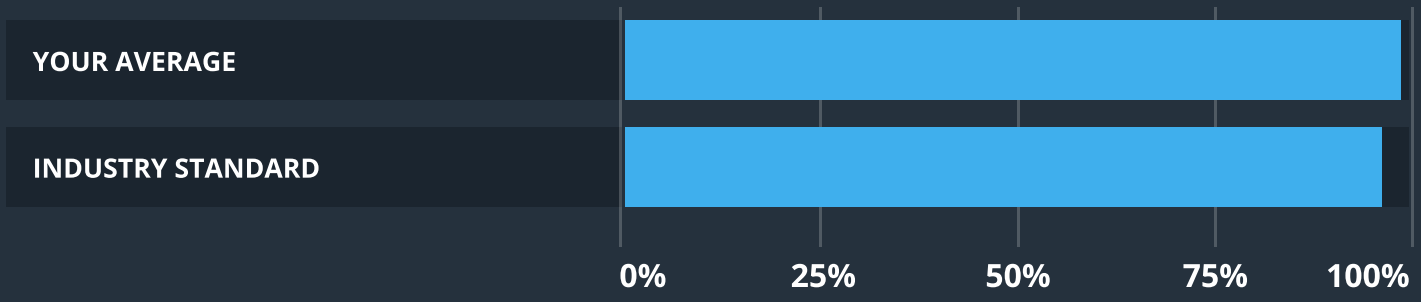
This document outlines the overall security of the FOX Yieldy smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the FOX Yieldy 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 98% 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 FOX Yieldy 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 3

Executive Summary 4

Structure and Organization of Document 5

Complete Analysis 6

Code Coverage and Test Results for all files 10

Code Coverage and Test Results for all files (2) 13

AUDITING STRATEGY AND TECHNIQUES APPLIED

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

Repository - https://github.com/shapeshift/foxy

Last commit - 2d0dd63ebdc40ff246c4f515170b0e7556c4f2d2

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

- FOX Yieldy.sol
- LiquidityReserve.sol
- Staking.sol
- Vesting.sol

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

- Implements and adheres to existing 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 resources, 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 FOX Yieldy 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. 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.

Contracts are well written and structured. The findings during the audit have no impact on contract performance or security, so it is fully production-ready.

Despite the fact, the expected logic is managing all vestings by the owner, it should be careful with parameters to avoid mistakes during the vesting process.

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. Issues tagged “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or it is an issue that the Customer disregards as an issue. 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

LOW | RESOLVED

In contract FOX Yieldy, function decreaseAllowance at lines 300-314, when allowed value is smaller then subtracted value will put 0 for allowed value. It can lead to undefined behavior.

Recommendation:
Add a require statement to check if _subtractedValue is smaller than allowed value.

LOW | RESOLVED

In contract Staking, the variable TOKE_REWARD_HASH, declared at line 23, and assigned at line 84, is not used after assignment.

Recommendation:
Remove the variable and constructor parameter.

INFORMATIONAL | RESOLVED

In contract FOX Yieldy, function rebase at lines 97-126, if circulating supply is 0 the call to _storeRebase will always revert. Move circulating supply check from _storeRebase at the beginning of function Rebase.

INFORMATIONAL | RESOLVED

In contract FOX Yieldy, function transfer at lines 210-220, should verify if sender has enough value.

INFORMATIONAL | RESOLVED

In contract FOX Yieldy, function transfer at lines 210-220, should verify if address of _to is not zero address.

INFORMATIONAL | RESOLVED

In contract FOX Yieldy, function transferFrom at lines 244-259, to prevent underflow panic, add a require for allowedValue[_from][msg.sender] - _value to be greater than 0. It's easier to understand when a custom message is provided.

INFORMATIONAL | RESOLVED

In contract LiquidityReserve, given that at line 11 the SafeERC20 is used for the IERC20 and in order to be consistent with other ERC20 transfer calls, use safeTransferFrom instead of transferFrom.

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

	LiquidityReserve.sol
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 FOX Yieldy team

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

Contract: **Foxy**
- Initialize
  - ✓ Should assign the total supply of tokens to the stakingContract
  - ✓ Fails if no stakingContract is passed to initialize (87ms)
- Rebase
  - ✓ Should distribute profits with one token holder (51ms)
  - ✓ Should distribute profits with two token holders (63ms)
  - ✓ Only can call rebase from staking contract
  - ✓ Rebase with no circulating supply
  - ✓ If profit = then no additonal funds should be received
- Approve
  - ✓ Sets the allowed value between sender and spender
  - ✓ Emits an Approval event
- increaseAllowance
  - ✓ Increases the allowance between sender and spender
  - ✓ Emits an Approval event
- decreaseAllowance
  - ✓ Decreases the allowance between sender and spender
  - ✓ Will not make the value negative
  - ✓ Emits an Approval event
- Integration
  - ✓ Should do everything (21792ms)

Contract: **Liquidity Reserve**
- deposit & withdraw
  - ✓ Should calculate the correct value of lrFOX with multiple providers (1810ms)
  - ✓ Liquidity providers should get correct amounts (1845ms)
  - ✓ Should not allow user to withdraw more than contract contains (1405ms)
- fail states

✓ Fails when no staking/reward token or staking contract is passed in (181ms)
✓ Must have correct fee amount
✓ Withdraw has required balance
✓ instantUnstake has required balance (303ms)

addLiquidity

✓ Issue correct balances of LR Fox with multiple LPs (2771ms)

Contract: Staking

Initialize

✓ Should assign the total supply of rewardToken to the stakingContract
✓ Fails when no staking/reward token or staking contract is passed in (59ms)

Stake

✓ User can stake, claim and unstake full amount when warmup period is 0 (702ms)
✓ Users have to wait for warmup period to claim and cooldown period to withdraw (1747ms)
✓ Fails to unstake when calling more than what user has in wallet or warmup contract (282ms)
✓ Users can unstake using funds from both wallet and warmup (1269ms)
✓ User can stake and unstake half amount without claiming (969ms)
✓ User can stake and unstake full amount without claiming (1062ms)
✓ Warmup period changing doesn't break stuff (1012ms)
✓ RequestedWithdrawals are 0 until sendWithdrawalRequests is called (769ms)
✓ Can instant unstake (1177ms)
✓ Can instant unstake without claiming (1300ms)
✓ User can stake and unstake multiple times with and without claiming (26829ms)
✓ Can't instant unstake if not enough liquidity reserve (296ms)
✓ when unstaking again without claimWithdraw it auto claims withdraw (1781ms)
✓ can unstake multiple times and get full amount (1478ms)
✓ unstakeAllFromTokemak allows users to unstake and claim rewards (1300ms)
✓ unstakeAllFromTokemak allows users to unstake and claim rewards with cooldown (1275ms)

reward

✓ Reward indexes are set correctly (644ms)
✓ Rewards can be added to contract and rebase rewards users (851ms)
✓ Unstakes correct amounts with rewards (1648ms)
✓ Gives the correct amount of rewards (5212ms)

Contract: Vesting

✓ Fails when no staking contract or reward token is passed in

sendWithdrawalRequest

✓ requestWithdrawalAmount is correct (26337ms')
✓ fails if either index isn't increased or batch period hasn't hit (14313ms)
✓ still sends if missed window (6051ms)

Tokemak

✓ Fails when incorrectly claims/transfer TOKE (1645ms)
✓ Staking gives tStakingToken to the Staking contract (207ms)
✓ Unstaking creates requestedWithdrawals (1289ms)

✓ Withdrawing gives the user their staking Token back from Tokemak (1065ms)
✓ Can't withdraw without first creating a withdrawRequest (323ms)
✓ Must wait for new index to send batched withdrawalRequests (6472ms)
✓ canBatchTransactions is handled appropriately (706ms)

Admin

✓ Admin functions work correctly (1290ms)
✓ Emergency exit is working (7336ms)

Ownable

✓ getOwner returns address that deployed contract
✓ can change owner and protects functions (52ms)

60 passing (3m)

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	Uncovered Lines
Staking	98.51	80	100	98.53	117
Vesting	100	94.44	100	100	
FOX Yieldy	100	100	100	100	
LiquidityReserve	100	100	100	100	
All files	**99.64**	**95.1**	**100**	**99.65**	

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

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

Contract: Staking

deployment
- ✓ should revert for zero address provided (245ms)
- ✓ should deploy and have correct epoch data (208ms)

owner
- ✓ should set onlyowner flags correctly (277ms)
- ✓ should not revert transfer Toke (182ms)
- ✓ should revert transfer Toke (205ms)

public funcs
- ✓ should revert stake for wrong input or paused stake (214ms)
- ✓ should transfer reward token if warmUpPeriod is zero (296ms)
- ✓ should create warmUpInfo then after expiry claim from warmUp vesting (435ms)
- ✓ should do nothing whem claim called but not available (172ms)
- ✓ should unstake from tokemak when called by owner (182ms)
- ✓ should revert claimFromTokemak for zero amount (205ms)
- ✓ should claim from claimFromTokemak (179ms)
- ✓ should send withdrawalRequest to tokemak (268ms)
- ✓ should not request withdrawal if can't batch txs (231ms)
- ✓ should not claim when claimWithdraw called after sendWithdrawalRequests (581ms)
- ✓ should revert unstake if unstake is paused (201ms)
- ✓ should unstake and send to cool down vesting (559ms)
- ✓ should unstake and trigger a rebase (551ms)
- ✓ should unstake and retrieve from warm up vesting (796ms)
- ✓ should revert instantUnstake if unstaking paused (217ms)
- ✓ should revert instantUnstake for zero reward tokens (271ms)
- ✓ should revert instantUnstake for not enough funds in reserve (422ms)
- ✓ should instantUnstake for enough liq reserve (1062ms)
- ✓ should revert unstake for insufficient funds (292ms)
- ✓ should not claimWithdraw when withdraw not available (282ms)
- ✓ should claimWithdraw when unstaking if withdraw available (774ms)
- ✓ should return from rebase when endBlock grater than block (195ms)

✓ should addRewardsForStakers and should not trigger rebase (309ms)
✓ should addRewardsForStakers and should trigger rebase (373ms)
✓ should instantUnstake when warmUpBalance is 0 (691ms)
✓ should instantUnstake when walletBalance is 0 (562ms)

Contract: Vesting

✓ should fail for no staking/reward token passed in
✓ should revert when not called by staking (48ms)
✓ should work when called by staking (53ms)

Contract: FOX Yieldy

initialize

✓ Should assign the total supply of tokens
✓ Verify if index is set
✓ Fails if 0x0 is passed to initialize (111ms)
✓ Fails if no stakingContract is passed to initialize (117ms)

rebase

✓ Caller is not StakingContract
✓ If profit is 0, total supply should remain the same
✓ Should revert when circulatingSupply = 0 and profit > 0
✓ Should modify total supply when circulatingSupply > 0 (87ms)
✓ TotalSupply must not exceed max supply (77ms)

approve

✓ Sets the allowed value to be spent
✓ Emits Approval event

increase Allowance

✓ Increases the allowed value to be spent when initial value = 0
✓ Increases the allowed value to be spent when initial value != 0 (47ms)
✓ Emits Approval event

decrease Allowance

✓ Decrease the allowed value to be spent when initial value = 0
✓ Decrease the allowed value to be spent when initial value != 0 and subtracted Value > old Value (81ms)
✓ Decrease the allowed value to be spent when initial value != 0 and subtracted Value < old Value (43ms)
✓ Decrease the allowed value to be spent when initial value != 0 and subtracted Value = old Value (80ms)
✓ Emits Approval event (39ms)

transferFrom

✓ Allowed value = 0 (38ms)
✓ Value to be transferred is higher than the allowed value (52ms)
✓ Value to be transferred is less than the allowed value (80ms)
✓ Emits Approval and Transfer events (70ms)

Contract: LiquidityReserve

```
deploy
  ✓ Invalid staking Token
initialize
  ✓ Invalid staking contract and reward Token (64ms
  ✓ Not enough staking tokens (176ms)
set Fee
  ✓ Must have correct fee amount
  ✓ Have correct fee amount
add Liquidity
  ✓ Add with success (500ms)
  ✓ Revert if liquiityReserve is not allowed to transfer (520ms)
remove Liquidity
  ✓ Not enough lr tokens
  ✓ Remove 100 (566ms)
  ✓ Not enough staking token (577ms)
instant Unstake
  ✓ Not from staking contract
  ✓ Remove 100 (514ms)
unstake All Reward Tokens
  ✓ Amount is 0
70 passing (51s)
```

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	Uncovered Lines
Staking	100	100	100	100	
Vesting	100	100	100	100	
FOX Yieldy	100	100	100	100	
LiquidityReserve	100	100	100	100	
All files	**100**	**100**	**100**	**100**	

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

ZOKYO.