TRUSTPAD

SMART CONTRACT AUDIT

ZOKYO.

Nov 1st, 2021 | v. 1.0

PASS

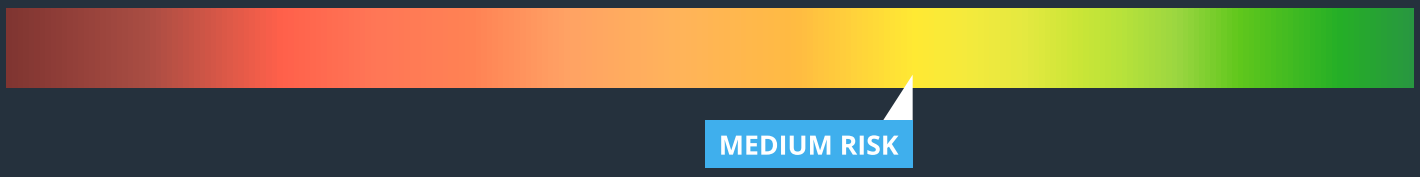
Zokyo Security Team has audited the given codebase. The overall score is presented in this section.

TECHNICAL SUMMARY

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

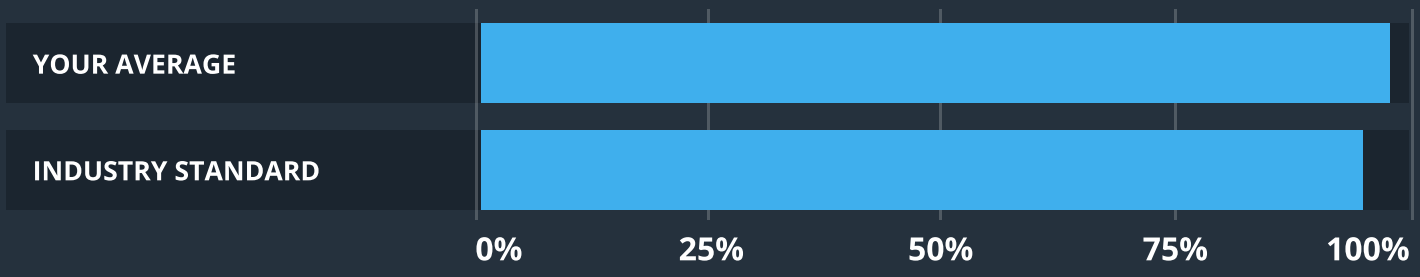
The scope of this audit was to analyze and document the TrustPad smart contract codebase for quality, security, and correctness.

Contract Status



There was 1 critical issue found during the audit.

Testable Code



The testable code is 97.97%, 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 TrustPad 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 TrustPad repository.

Repository:
https://github.com/trustpad/contracts-audit/commit/842cdec809d13e3f41605cf5597bde62e812d68a

Last commit:
8e1ce8b

Contracts under the scope:

- LaunchpadStaking;
- Stakeable;
- StakingTreasury;
- ILevelManager;
- IStakingLockable;
- LevelManager;
- WithLevels;
- WithPools;
- Claimer;
- GeneralIDO;
- LaunchpadDO;
- Timed;
- Withdrawable;
- WithLevelsSale;
- WithLimits;
- WithWhitelist.

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 TrustPad 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.

SUMMARY

The Zokyo team has concluded a security audit for the given smart contracts. During the process, a couple of issues were found. All of the mentioned issues are described below in the Compete Analysis section.

The auditor's team found 1 critical and a couple of issues with a low level of severity. It is worth mentioning that the critical issue was not resolved and the contracts are not compiled. (ast commit - 8e1ce8b is not compiled).

Based on the list of issues and the quality of the codebase we can state that the contracts should be double-checked by the Trustpad team. We insist on fixing critical issue as it is not resolved.

The overall score of the audited smart contracts is 86.

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 ability of the contract to compile or operate in a significant way.

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

The contract WithPools.sol is not compiled

CRITICAL | UNRESOLVED

In the WithPools.sol contract, the types for comparison do not match in the poolExists function at line 51, because pools is an array of type IStakingLockableExternal, and the pool is a regular address.

Recommendation:
Make a pool of type IStakingLockableExternal, example:

```
if(pools[i] == IStakingLockableExternal(pool))
```

Re-audit:
Skipped.

Unreachable code

LOW | UNRESOLVED

In Steakable.sol file, in deposit(), the if(user.amount==0) will never get satisfied because the minimum value that will get updated in the user.amount variable before is Non-zero value.

Recommendation:
Do user.amount calculations unconditionally amount > 0, then stakersNumber will be increasing.

Re-audit:
Skipped.

Unreachable code

LOW | UNRESOLVED

In Steakable.sol file, in withdraw(), the if(user.amount == 0 && stakersNumber > 0) will never get satisfied because stackersNumber will not be added when deposited.

Recommendation:
Fix issue in function deposit() for increase stakersNumber.

Re-audit:
Skipped.

Lock pragma to a specific version

LOW | RESOLVED

Lock the pragma to a specific version, since not all the EVM compiler versions support all the features, especially the latest one's which are kind of beta versions, So the intended behavior written in code might not be executed as expected.

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs.

Re-audit:
Resolved.

Few checks in the required statements and suggestions

LOW | RESOLVED

1) In withPools.sol file, in addPool() check can be made to make sure that the pool is not added already;
2) In withPools.sol file, in setPoolMultiplier(), check whether the pool is present before updating the multiplier attribute;
3) In Claimer.sol file, in withdrawAll(), before 188 line check whether token is not zero, that is:

```
require(address(token) != address(0), "Claimer: token address is not present")
```

4) In LaunchPad.sol, in internalBuyTokens() function, move the line:

```
equire(amount > 0, "Sale: amount is 0");
```

before the line 85, that is before updating the tokenSold variable;

5) In LevelManager.sol, in getUserAmount(), in line 45, check the amount is whether a valid number or not before doing the calculation;
6) In withdrawable.sol, in withdrawBalance() & withdrawToken() functions, check should be made before withdrawing the amount, that is required:

```
(amount >0, "Withdrawable: Amount should be greater than zero")
```

7) In Stakeable.sol, in updatePool() function, in line 80, no need of if condition because it will never get satisfied except when:

```
blocknumber=liquidityMining.lastRewardBlock
```

8) Use math.sol for all arithmetic operations wherever possible.

Re-audit:
Resolved.

	LaunchpadStaking; Stakeable; StakingTreasury; ILevelManager; IStakingLockable; LevelManager; WithLevels; WithPools;	Claimer; GeneralIDO; LaunchpadDO; Timed.; Withdrawable; WithLevelsSale; WithLimits; WithWhitelist.
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 Security team

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

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	100.00	100.00	100.00	100.00	
Adminable.sol	100.00	100.00	100.00	100.00	
contracts\claim\	100.00	100.00	100.00	100.00	
Claimer.sol	100.00	100.00	100.00	100.00	
contracts\ido\	94.97	86.46	100.00	94.63	
GeneralIDO.sol	100.00	100.00	100.00	100.00	
LaunchpadDO.sol	97.92	89.47	100.00	98.00	147
Timed.sol	100.00	91.67	100.00	100.00	
WithLevelsSale.sol	89.02	69.23	100.00	88.10	... 101, 104, 214
WithLimits.sol	100.00	100.00	100.00	100.00	
WithWhitelist.sol	100.00	100.00	100.00	100.00	
Withdrawable.sol	100.00	100.00	100.00	100.00	
contracts\levels\	97.73	90.63	100.00	96.74	
ILevelManager.sol	100.00	100.00	100.00	100.00	

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
IStakingLockable.sol	100.00	100.00	100.00	100.00	
LevelManager.sol	94.44	78.57	100.00	92.31	50, 51, 52
WithLevels.sol	100.00	100.00	100.00	100.00	
WithPools.sol	100.00	100.00	100.00	100.00	
contracts\staking\	97.14	90.38	100.00	97.20	
LaunchpadStaking.sol	100.00	100.00	100.00	100.00	
Stakeable.sol	96.70	89.58	100.00	96.70	83, 120, 150
StakingTreasury.sol	100.00	100.00	100.00	100.00	
All files	**97.97**	**93.49**	**100.00**	**97.71**	

Test Results

Contract: Claimer

Claimer Initialize Phase Test Cases

- ✓ should initialize token correctly (152ms)
- ✓ should initialize id correctly (160ms)
- ✓ should add percent correctly (97ms)
- ✓ shouldn't initialize if the sum of all claimed isn't 100% (757ms)
- ✓ shouldn't initialize if percents are zer (316ms)
- ✓ shouldn't initialize if times is zero (333ms)

Claimer Functions Phase Test Cases

- ✓ should set token correctly (851ms)
- ✓ should withdraw tokens correctly (616ms)
- ✓ should withdraw all tokens correctly (553ms)
- ✓ should withdraw unexpected ethers correctly (477ms)
- ✓ should claim correctly (687ms)
- ✓ shouldn't claim if out of bounds index (307ms)
- ✓ shouldn't claim if already claimed (695ms)
- ✓ shouldn't claim if account doesn't have allocation (305ms)
- ✓ shouldn't claim if account doesn't have allocation (593ms)

✓ should set pause claiming correctly (225ms)
✓ shouldn't claim if not claimable (380ms)
✓ shouldn't claim if amount is zero (610ms)
✓ should get account's amount correctly (343ms)
✓ should release claim correctly (275ms)
✓ shouldn't release claim if out of bounds index (326ms)
✓ shouldn't release claim if claim already released (367ms)
✓ should delay claiming correctly (292ms)
✓ shouldn't delay claim if out of bounds index (291ms)
✓ shouldn't delay claim if new timeless than timestamp (342ms)
✓ shouldn't delay claim if new timeless than current claim time (274ms)
✓ should get info about account amount correctly (884ms)
✓ should get info about account amount correctly if isClaimable is false (893ms)
✓ should batch add allocation correctly (523ms)
✓ should batch mark claimed correctly (580ms)
✓ should get claimes correctly (1656ms)
✓ should get remaining correctly (1197ms)

Contract: LaunchpadStaking

LaunchpadStaking Functions Phase Test Cases

✓ should set level manager correctly (223ms)
✓ shouldn't set level manager if msg.sender isn't owner or admin (603ms)
✓ should set halt correctly (422ms)
✓ should add deposit correctly (905ms)
✓ shouldn't add deposit if deposits are paused (586ms)
✓ should withdraw rewards correctly (904ms)
✓ should withdraw correctly (2381ms)
✓ shouldn't withdraw if account of caller is locked (2584ms)

Stakeable Functions Phase Test Cases

✓ shouldn't set pool info if token is already set (259ms)
✓ shouldn't set start mining if mining already started (359ms)
✓ should set fee correctly (506ms)
✓ shouldn't set fee if new fee more than 50 (286ms)
✓ should set reward perBlock correctly (413ms)
✓ shouldn't set reward perBlock if new reward perBlock is zero (196ms)
✓ should withdraw fees correctly (1393ms)
✓ shouldn't claim if not enough reward tokens for transfer (889ms)
✓ should claim correctly if pending rewards of user are zero (681ms)

✓ shouldn't claim if not enough reward tokens for transfer (1529ms)
✓ should claim also user.pendingRewards correctly (1568ms)
✓ should claim also the balance of treasury correctly (1513ms)
✓ shouldn't withdraw if withdrawing more than you have (623ms)
✓ shouldn't do deposit if mining not yet started (603ms)
✓ should get pending rewards correctly (2546ms)
✓ should update pool correctly if the balance of staking token is zero (1527ms)
✓ should withdraw amount with fee correctly (2222ms)
✓ should withdraw amount correctly if user.amount is zero (680ms)

Contract: LevelManager

LevelManager Functions Phase Test Cases

✓ shouldn't add IDO if account has zero's address (600ms)
✓ should toggle locking correctly (366ms)
✓ shouldn't lock account if ido start less than timestamp (324ms)
✓ shouldn't lock account if the caller isn't IDO (299ms)
✓ should unlock account correctly (624ms)
✓ should batch lock accounts correctly (1080ms)

WithLevels Functions Phase Test Cases

✓ should create tier correctly (1371ms)
✓ shouldn't set tier if id is none (209ms)
✓ should update tier correctly (1382ms)
✓ should delete tier correctly (1845ms)
✓ shouldn't delete tier if id is none (337ms)
✓ should get tier by id correctly (1614ms)
✓ should get tier idx for amount correctly (1852ms)

WithPools Functions Phase Test Cases

✓ should add pool correctly (648ms)
✓ should set pool multiplier if the multiplier is zero (325ms)

Contract: LaunchpadIDO

Withdrawable Functions Phase Test Cases

✓ shouldn't add address of fundtoken if zero's address (179ms)
✓ should set fundtoken correctly (744ms)
✓ should withdraw token correctly (658ms)
✓ should withdraw balance correctly (508ms)
✓ should withdraw balance with unexpected ether correctly (9056ms)
✓ should withdraw all balance correctly (6241ms)
✓ should withdraw all balance with unexpected ether correctly (8780ms)

WithWhitelist Functions Phase Test Cases
- ✓ should toggle whitelist correctly (351ms)
- ✓ should batch add whitelisted correctly (1332ms)
- ✓ should batch remove whitelisted correctly (1291ms)

WithLimits Functions Phase Test Cases
- ✓ should set min correctly (485ms)
- ✓ should set max correctly (452ms)
- ✓ shouldn't set min if min more than max (829ms)
- ✓ shouldn't set max if max less than min (618ms)

Timed Functions Phase Test Cases
- ✓ should set start time correctly (655ms)
- ✓ shouldn't set start time if new start time less than register time (582ms)
- ✓ should set duration correctly (365ms)
- ✓ should set register time correctly (513ms)
- ✓ shouldn't set register time if new register time more than start time (268ms)
- ✓ should set register duration correctly (365ms)
- ✓ shouldn't set register duration if amount of register time and duration more than start time (296ms)
- ✓ should set FCFS duration correctly (459ms)
- ✓ should set timeline correctly (1095ms)

LaunchpadIDO Functions Phase Test Cases
- ✓ should get the message from receive function correctly (376ms)
- ✓ should get receive function correctly (8858ms)
- ✓ should get fallback function correctly (275ms)
- ✓ should buy tokens correctly (8169ms)
- ✓ shouldn't buy tokens if funding by tokens is not allowed (5157ms)
- ✓ shouldn't buy tokens if fund token not approved (5334ms)
- ✓ shouldn't buy tokens if the base allocation isn't set (6609ms)
- ✓ shouldn't buy tokens if amount is too small (11304ms)
- ✓ shouldn't buy tokens if amount exceeds max allocation (10593ms)
- ✓ shouldn't buy tokens if cap reached (13089ms)
- ✓ shouldn't buy tokens if not approved (910ms)
- ✓ shouldn't buy tokens if isLive is false (791ms)
- ✓ shouldn't buy tokens if the amount is zero (9633ms)
- ✓ shouldn't buy tokens if not in the whitelist (8687ms)
- ✓ shouldn't buy tokens if user level isn't registered (3734ms)
- ✓ should check account allowed to buy correctly (12210ms)
- ✓ should check account allowed to buy if not opened for all levels (9933ms)

```
  WithLevelsSale Functions Phase Test Cases
    ✓ shouldn't register if not open for registration (427ms)
    ✓ shouldn't register if levels staking address is not specified (1025ms)
    ✓ shouldn't register if a level is too low to register (3639ms)
    ✓ shouldn't register account if caller isn't IDO (5227ms)
    ✓ should register account correctly (6098ms)
    ✓ shouldn't register account if already registered with lower level (7400ms)
    ✓ shouldn't buy tokens if levels are disabled (1576ms)
    ✓ should get level addresses correctly (1189ms)
    ✓ should get level numbers correctly (1811ms)
    ✓ should toggle lock on register correctly (506ms)
    ✓ should set min allowed level multiplier correctly (571ms)
    ✓ should set winners correctly (6663ms)
    ✓ should update base allocation correctly (7676ms)
    ✓ should get fcfs allocation multiplier correctly (713ms)
  GeneralIDO Functions Phase Test Cases
    ✓ should set tokens for sale correctly (606ms)
    ✓ shouldn't set tokens for sale if a sale is live (725ms)
    ✓ should set rate correctly (498ms)
    ✓ shouldn't set rate if a sale is live (858ms)

  131 passing (16m)
```

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

ZOKYO.