Pontoon

SMART CONTRACT AUDIT

ZOKYO.

September 10th, 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

SCORE

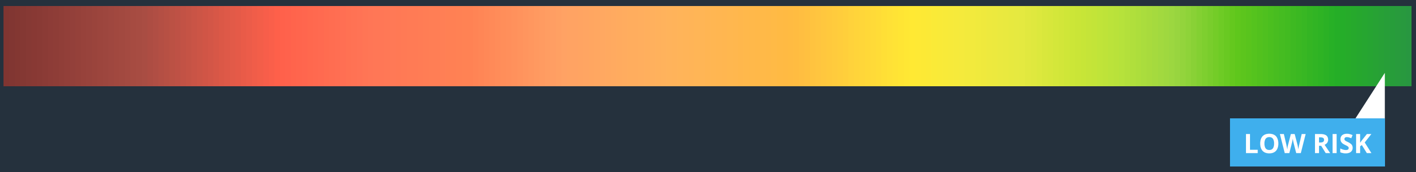
100

TECHNICAL SUMMARY

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

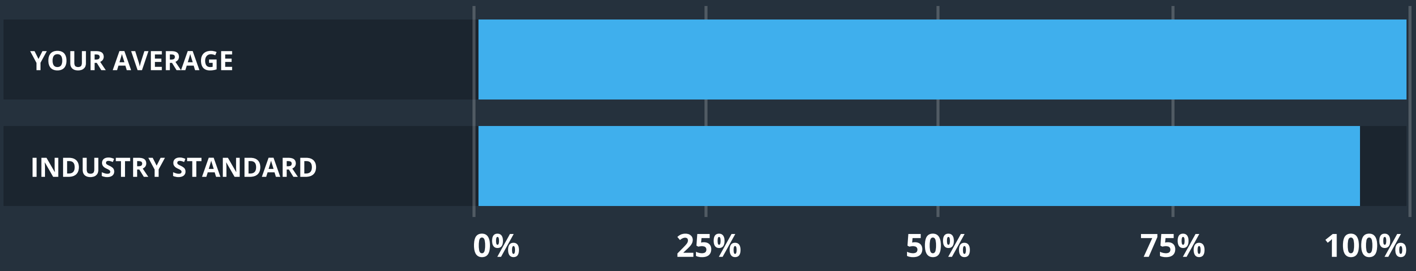
The scope of this audit was to analyze and document the Pontoon smart contract codebase for quality, security, and correctness.

Contract Status



There were no critical issues found during the audit.

Testable Code



The testable code is 100%, 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 Pontoon 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 Pontoon repository.

Repository:
https://github.com/pontoonfi/pontoon-staking

Last commit:
85f8fa3a461b949b784de369dafbfc500046d0fb

Contracts:

- PontoonFarm.sol
- PontoonFarmFactory.sol
- Utils.sol
- Vault.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 Pontoon 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.
2. Cross-comparison with other, similar smart contracts by industry leaders.
3. Testing contract logic against common and uncommon attack vectors.
4. Thorough, manual review of the codebase, line-by-line.

SUMMARY

Zokyo auditing team has conducted a smart contract security audit. During the process, no major issues have been identified. Zokyo auditors came up with only 1 objection, marked as an informational issue. This finding was successfully resolved by the Pontoon team.

We can state, that the contracts are in excellent condition. Based on the conducted audit, we can give a score of 100 to the smart contracts under the scope.

Zokyo security team confirms that the codebase provided for this audit bears no security or operational risk and is fully production-ready.

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

Too low comments coverage

INFORMATIONAL | RESOLVED

Most of the functions and contracts are not described in the comments. It means that there can be misunderstandings between different developers. To avoid this solidity, support NatSpec format of comments. Also, contracts in Utils do have comments but not in NatSpec format.

Recommendation:
Make comments for main functionality in NatSpec format.

Re-audit:
Fixed.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Pontoon team

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\	73.60	72.73	73.33	73.79	
PomtoonFarm.sol	80.18	72.73	81.82	79.82	... 279, 280, 281
PontoonFarmFactory.sol	0.00	100.00	0.00	0.00	... 48, 50, 52, 56
Utils.sol	100.00	100.00	100.00	100.00	
Vault.sol	100.00	100.00	100.00	100.00	
contracts\mocks\	100.00	100.00	100.00	100.00	
BEP20.sol	100.00	100.00	100.00	100.00	
IBEP20.sol	100.00	100.00	100.00	100.00	
All files	**73.81**	**72.73**	**76.47**	**73.60**	

Test Results

```
PontoonFarm: emergencyUnstake
  ✓ should return all the user's balance back to the user (149ms)
  ✓ should not transfer pending rewards to the user (145ms)
  ✓ should update user info (117ms)
  ✓ should return all the user's balance back to the user (123ms)

PontoonFarm: init
  ✓ should revert if array lengths do not match
```

```
    ✓ should revert if claim duration is larger than block duration
    ✓ should revert if vault is not a contract
    ✓ should revert if staking token is not a contract (56ms)
    ✓ should revert if any of the reward tokens is not a contract
    ✓ should set the initial values correctly (102ms)

  PontoonFarm: reward calculations
    ✓ should calculate rewards correctly with one user (337ms)
    ✓ should calculate rewards correctly with more than one users (620ms)
    ✓ should calculate rewards correctly with one user and one reward token (283ms)
    ✓ should calculate rewards correctly with more than one users and one reward token (476ms)
    ✓ should calculate rewards correctly with more than one users and one 4 reward tokens (742ms)

  PontoonFarm: stake
    ✓ should send pending rewards to the user (147ms)
    ✓ should get the staking tokens from the user (64ms)
    ✓ should update the users info (97ms)
    ✓ should update the users info after getting rewards (113ms)
    ✓ should update the users accClaims (283ms)
    ✓ should emit a stake event (58ms)

  PontoonFarm: unstake
    ✓ should return the unstake amount back to the user (123ms)
    ✓ should revert if trying to unstake more than have been staked (122ms)
    ✓ should transfer pending rewards to the user (127ms)
    ✓ should emit an UnStaked event (122ms)
    ✓ should update user info (132ms)

  26 passing (15s)
```

Tests written by Zokyo Security team

As part of our work assisting Pontoon 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 Pontoon 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	
PontoonFarm.sol	100.00	100.00	100.00	100.00	
PontoonFarmFactory.sol	100.00	100.00	100.00	100.00	
Utils.sol	100.00	100.00	100.00	100.00	
Vault.sol	100.00	100.00	100.00	100.00	
All files	**100.00**	**100.00**	**100.00**	**100.00**	

Test Results

```
Contract: PontoonFarmFactory
  Initialization
    ✓ should deploy correctly (3535ms)

Contract: PontoonFarm
  Initialization
    ✓ cannot deploy if the length of the arrays does not match (1536ms)
    ✓ cannot deploy if claimDelayDuration more than blockDuration (392ms)
    ✓ cannot deploy with more than four reward tokens (387ms)
```

```
      ✓ cannot deploy with invalid vault address (465ms)
      ✓ cannot deploy with more than four reward tokens (387ms)
      ✓ cannot deploy with invalid rewardToken address (434ms)
    Stake
      ✓ should stake correctly (1430ms)
      ✓ cannot stake if farming has not started (849ms)
      ✓ cannot stake if farming has ended (901ms)
    Unstake
      ✓ should unstake correctly (1332ms)
      ✓ cannot unstake more than staked (303ms)
    UserData
      ✓ should return user data correctly (190ms)
    PendingRewards
      ✓ should return pending rewards correctly (884ms)
    EmergencyUnstake
      ✓ should unstake all rewards correctly (946ms)
    WithdrawRemainingRewards
      ✓ should withdraw remaining rewards correctly (1446ms)
      ✓ cannot withdraw remaining rewards if farming has started (180ms)
    ReleaseRewards
      ✓ should release rewards correctly if block.number will be bigger than claimDelayBlock
        and totalPending will be bigger than zero (9645ms)
      ✓ should release rewards correctly (7852ms)

  Contract: Utils
    isContract
      ✓ should correctly recognize the address of the contract (373ms)

  Contract: Vault
    SafeRewardTransfer
      ✓ should transfer rewards correctly (500ms)

  21 passing (43s)
```

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

ZOKYO.