SMART CONTRACT AUDIT

ZOKYO.

June 11th, 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

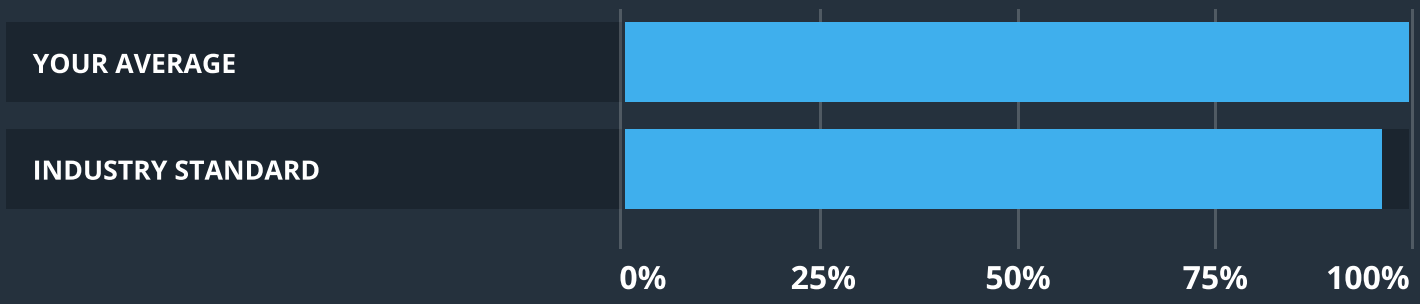
This document outlines the overall security of the Pontoon smart contracts, evaluated by Zokyo's Blockchain Security team.

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 Bridge repository.

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

Commit: e23ddc9a423e749c93dd1679a6257daa5d18d227
Last commit: 6a8df4f59599e1568fda97a77e1468a9224f91fb

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.	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 was found an unclear functionality and mistakes in storage parameters usage. Though, all functionality was verified by the Pontoon team. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner.

The findings during the audit have a slight impact on contract performance and code style.

All issues were successfully fixed and verified by the Pontoon team.

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

Unsafe tx tracking

PontoonBridge.sol, redeem(), swap()

Functions get txNumber as an argument, so it is generated off-chain. In general it is not a good practice, because it leaves the place for potentially malicious behavior, duplicating transactions, incorrect ordering or even double spending. So, the best practice is to generate txNumber on-chain (as additional counter) or at least to store txNumbers in a separate array and mappings. Thus it will be possible to track the order of transaction, ensure that the tx is unique and has not been performed yet.

Recommendation:

Add mapping to track status of txNumber (to ensure that it was unique) and/or array to have the order of txNumbers and/or generate txNumber on-chain.

Post-audit: marked as resolved after the conversation with Pontoon team

HIGH | RESOLVED

Incorrect fee check

PontoonBridge.sol, line 433

Fee (parameter "_fee") is not checked before the update. The code checks the storage parameter "fee" - "require(fee <= (10**feePrecision)". So, any value can be set at least once.

Recommendation:

Use the function's parameter, not the storage one.

MEDIUM | RESOLVED

No ability to change tokens pool

PontoonBridge.sol
The bridge has no ability to change the pool in case if that becomes compromised or becomes updated. Consider the owner only method to change the bridge with liquidity migration or confirm the functionality.

Recommendation:
Add owner only ability to change the pool, or confirm the functionality.

LOW | RESOLVED

Magic number for the accuracy and lack of checks

PontoonBridge.sol, line 239
PontoonBridge.sol, updateFee(), line385
Accuracy for the fee calculation is set to the “magic number”, so in order to increase code quality consider moving it to the constant. Also, function updateFee() has a lack of checks against the accuracy - that fee cannot exceed the accuracy.

Recommendation:
Use constant for the accuracy and add check for the fee update.

LOW | RESOLVED

Missing explicit visibility

PontoonBridge.sol, line 44
liquidityPools variable has no explicit visibility.

Recommendation:
Add visibility mark.

LOW | RESOLVED

Low risk reentrancy

PontoonBridge.sol, redeem(), swap()
In spite of the fact that tokens to bridge are added by the owner, that bridge checks the status of the swap, existence of the pool for tokens, tokens themselves cannot be validated completely. Thus, there is a low level risk of reentrancy to be performed against the exploited overloaded “safeTransferFrom()” method. So it will be better to perform the protection against such kind of attack.

Recommendation:
Consider usage of the standard ReentrancyGuard for redeem() and swap() methods.

LOW | RESOLVED

Functions should be declared as external
In order to get gas savings and increase the overall security it is recommended to declare external functions:
PontoonBridge.sol #112, 189, 347, 385:
swap(address,string,uint256,address,address,uint8)
redeem(address,address,string,uint256,address,address,uint8,uint8,bytes32,bytes32)
updateValidator(address)
updateFee(uint256)
PontoonPool.sol 66, 80, 137, 144:
transferLiquidity(address,uint256)
addLiquidity(uint256)
removeLiquidity(uint256)
updateAccruedFee(uint256)
getUnlockTime()
updateLockPeriod(uint256)
ECDSA.sol 11:
getSigner(bytes32,uint8,bytes32,bytes32)
Misk.sol 5:
isContract(address)

Recommendation:
Declare functions as external.

INFORMATIONAL | RESOLVED

Check sender and recipient address

The check is needed to prevent misleading transactions or a transaction with incorrect arguments (e.g. because of frontend validation failure).
It will decrease overall number of transactions with no effects or with no positive effects
PontoonBridge.sol #189:
redeem(address, address, string memory, uint256, address, address, uint8, uint8, bytes32, bytes32)
PontoonPool.sol #66:
transferLiquidity(address, uint256)

Recommendation:
Add “require” statement to prevent equality of sender and recipient
Post-audit: marked as resolved after the conversation with Pontoon team

INFORMATIONAL | RESOLVED

Lacks a zero check-on

Detect missing zero address validation in functions:
PontoonBridge.sol #347:
updateValidator(address)
PontoonPool.sol #37:
constructor(uint256,address,string,string,address)

Recommendation:
Check that the address is not zero.

	PontoonBridge	PontoonPool
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 Pontoon 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 Pontoon contract requirements for details about issuance amounts and how the system handles these.

```
Test Pontoon
  Deploy contracts
    √ STEP 1: Should create the Tokens (7754ms)
  PontoonBridge: ctor
    √ STEP 1: Should deploy the bridges (1871ms)
    √ STEP 2: Should store the validator address (167ms)
    √ STEP 3: Should store the fee (175ms)
    √ STEP 4: Should store the lockPeriod (187ms)
  PontoonPool: addPool
    √ STEP 1: Should revert if not called by the owner (2218ms)
    √ STEP 2: Should revert if given token parameter is not a contract address (1235ms)
    √ STEP 3: Should deploy a new PontoonPool with the correct params (11218ms)
    √ STEP 4: Should revert there is already a pool for the given token (627ms)
    √ STEP 5: Should store the address of the newly deployed pool (278ms)
    √ STEP 6: Should store the token address in the list of token addresses (136ms)
  PontoonPool: ctor
    √ STEP 1: Should store the lockPeriod (142ms)
    √ STEP 2: Should store the token address (254ms)
    √ STEP 3: Should store the bridge address (143ms)
    √ STEP 4: Should set the normalization factor to 0 if the given token has 18 decimals (127ms)
    √ STEP 1: Should set the correct normalization factor  the given token has fewer than 18 decimals (162ms)
  PontoonPool: addLiquidity
    √ STEP 1: Should calculate the amount of LP tokens when the underlying source token has 18 decimal points (1469ms)      √
STEP 2: Should calculate the amount of LP tokens when the underlying source token has fewer than 18 decimal points (3368ms)
    √ STEP 3: Should update the sender's unlock time to the correct date in future (174ms)
    √ STEP 4: Should transfer the given amount from the sender to the pool address (138ms)
  PontoonBridge: swap
    √ STEP 1: Should revert if there is no pool registered for the src token (489ms)
    √ STEP 2: Should transfer the given amount of src tokens from the recipient to the designated pool contract (2744ms)
    √ STEP 3: Should revert if swap is not in an Empty state (482ms)
    √ STEP 4: Should store all swap related data in the swaps mapping (285ms)
```

```
  PontoonBridge: redeem
    √ STEP 1: Should revert if the provided message was not signed by the validator (738ms)
    √ STEP 2: Should revert if swap is not in an Empty state (472ms)
    √ STEP 3: Should revert if there is no liquidity pool for the given destination tokens (652ms)
    √ STEP 4: Should revert if there is no liquidity in the liquidity pool (3755ms)
    √ STEP 6: Should set the state of the swap to Redeemed (822ms)
    √ STEP 7: Should calculate and update the accrued fees on the liquidity pool (138ms)
    √ STEP 8: Should transfer the amount of tokens minus the fee from the liquidity pool to the recipient address (320ms)
    √ STEP 9: Should seamlessly work even if the fee is 0 (3307ms)
  PontoonPool: removeLiquidity
    √ Step 1: Should revert if sender does not have the correct balance of lp tokens (374ms)
    √ Step 2: Should revert if sender lp token balance is smaller than the given amount (466ms)
    √ Step 3: Should revert if unlock time has not passed (409ms)
    √ Step 4: Should transfer the given amount of the underlying token plus accrued fees (factor=0) (3015ms)
    √ Step 5: Should transfer the given amount of the underlying token plus accrued fees (factor!=0) (2273ms)
    √ Step 6: Should update the accruedFee field (factor=0) (135ms)
    √ Step 7: Should update the accruedFee field (factor!=0) (143ms)
    √ Step 8: Should update the sender's lpTokenBalance (162ms)
    √ Step 9: Should burn the given amount of lp tokens (176ms)
  PontoonPool: updateAccruedFee
    √ STEP 1: Should revert if the not called by the bridge address (341ms)
    √ STEP 2: Should calculate and update the accrued fees on the liquidity pool (1180ms)
  PontoonPool: transferLiquidity
    √ STEP 1: Should revert if the not called by the bridge address (270ms)
    √ STEP 2: Should transfer the amount of tokens minus the fee from the liquidity pool to the recipient address (1252ms)
  PontoonPool: all other methods
    √ STEP 1: Should return the unlock time for the given address (1284ms)
    √ STEP 2: Should revert if not called by the owner (660ms)
    √ STEP 3: Should update the lock period value (1268ms)
```

```
  PontoonBridge: updatePool
    √ STEP 1: Should revert if not called by the owner (321ms)
    √ STEP 2: Should revert if given token parameter is not a contract address (291ms)
    √ STEP 3: Should deploy a new PontoonPool with the correct params (6663ms)
    √ STEP 4: Should revert there is no pool for the given token (435ms)
    √ STEP 5: Should store the address of the newly deployed pool (186ms)
    √ STEP 6: Should store the token address in the list of token addresses (183ms)
  Test other functionalities
    √ Should revert in constructor if validator is zero address (702ms)
    √ Should not update validator if new validator is zero address (378ms)
    √ Should set new validator (488ms)
    √ Should update lock period (414ms)
    √ Should not update fee if it is larger than precision (258ms)

Test Pool
  Testing constructor
    √ Should revert token address is zero (534ms)
    √ Should revert if bridge address is zero (688ms)
  Testing luquidity
    √ Should add liquidity (986ms)
    √ Should not remove liquidity if user is not provider (473ms)
    √ Should not remove liquidity if not time yet (409ms)
    √ Should not remove liquidity if amount exceeds balance (671ms)
    √ Should remove liquidity (961ms)
    √ Should revert transfer liquidity if recipient is zero address (242ms)
    √ Should revert if caller is not a bridge (264ms)
    √ Should transfer liquidity (818ms)
  Test other functions
    √ Should update fee (427ms)
    √ Should get unlock time (178ms)
    √ Should update unlock period (374ms)
```

```
 72 passing (1m)

--------------------|----------|----------|----------|----------|----------------|
File                |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
--------------------|----------|----------|----------|----------|----------------|
 contracts\         |      100 |      100 |      100 |      100 |                |
  PontoonBridge.sol |      100 |      100 |      100 |      100 |                |
  PontoonPool.sol   |      100 |      100 |      100 |      100 |                |
--------------------|----------|----------|----------|----------|----------------|
All files           |      100 |      100 |      100 |      100 |                |
--------------------|----------|----------|----------|----------|----------------|
```

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.