Security Audit Report for PontoonFi Contracts

Date: Sep 27, 2021

Version: 1.0

Contact: contact@blocksecteam.com

Contents

Report Manifest

Item	Description
Client	Skeeve Labs Ltd.
Target	PontoonFi Contracts

Version History

Version	Date	Description
1.0	September 27, 2021	First Release

About BlockSec The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and released detailed analysis reports of high impact security incidents. The team won first place in the 2019 iDash competition (SGX Track). They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Pontoon is a cross-chain AMM – bridge between different chains. It's a decentralized application based on smart contracts. Users can exchange coins between different blockchains.

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The smart contracts that have been audited in this report include the following ones. We show both the commit hashes before and after this audit.

Before

Project	Commit SHA
pontoon	6a8df4f59599e1568fda97a77e1468a9224f91fb
pontoon-staking	c08dbe7955e154a0789bab8ede7c25f4228d5a30
pontoon-token	66189b68458ed17940b8db33d5a5b3fcb1213361

After

Project	Commit SHA
pontoon	803cf2a0a80748258a2ddd2c4bdb7056a24b5839
pontoon-staking	91052d12debb5905305efdb96fcae699489dd082
pontoon-token	2fb6f3a28c0eaed943d0fb628cb3507a6b19fe13

1.2 Disclaimer

This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, this report does not constitute any personal investment advice or personal recommendation.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.

- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data Flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Access control
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [1] and Common Weakness Enumeration [2]. Accordingly, the severity measured in this report are classified into four categories: **High**, **Medium**, **Low** and **Undetermined**.

[1] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[2] https://cwe.mitre.org/

Chapter 2 Findings

In total, we have identified **12 potential issues** and 4 additional recommendations, as follows:

- High Risk: 2
- Medium Risk: 3
- Low Risk: 7

ID	Severity	Description	Category
1	High	Reentrancy	Software Security
2	Medium	Unchecked `_destChainId`	Software Security
3	Low	Unchecked Parameters	Software Security
4	High	Incorrect Implementation of `safeRewardTransfer()`	Software Security
5	Medium	Neglection of Deflation & Inflation Token	DeFi Security
6	Medium	Neglection of Token with Non-Standard Decimals	DeFi Security
7	Low	Unchecked Equivalence of Token Decimals	DeFi Security
8	Low	No Cancellation Mechanism	DeFi Security
9	Low	No Pausing Mechanism	DeFi Security
10	Low	Inconsistent Function Behaviors	DeFi Security
11	Low	Transfer Tokens without Invoking the `stake()` Function	DeFi Security
12	Low	Lock Period Bypass	DeFi Security
13		Single Validator	Additional Recommendations
14		Better Practice for Management of Staked Tokens	Additional Recommendations
15		Better Function Return Value	Additional Recommendations
16		Misspelling in Comments	Additional Recommendations

The details are provided in the following sections.

2.1 Software Security

2.1.1 Reentrancy

Status Confirmed and fixed.

Description

In PontoonBridge, there is a potential reentrancy vulnerability, which can be attacked as follows:

1) Suppose the `_srcToken` is a token that supports the callback mechanism (like ERC-777 token). The invocation of `swap()` in PontoonBridge will invoke `safeTransferFrom()`, which then calls **user(attacker)-controlled** callback functions.

2) The callback function then calls `liquidityPools[_srcToken].addLiquidity()` (in PontoonPool). The attacker will receive the LP token of the added liquidity.
3) Then when the function call returns back to PontoonBridge, the `balanceOf(address(liquidityPools[_srcToken])` will be much larger and the `netAmount` is miscalculated (larger than expected).

Similarly, the reentrancy vulnerability could be exploited in the reverse direction in PontoonPool:

1) Suppose `_srcToken` supports callbacks, the invocation of `addLiquidity()` in PontoonPool will call `safeTransferFrom()` and then call the user-defined callback function.
2) In the callback function, call `swap()` in PontoonBridge and PontoonBridge will transfer `_srcToken` to the pool.
3) When returning back to PontoonPool, the `netAmount` will be much larger because `swap()` in PontoonBridge changed the balance of PontoonPool.

```
122  function swap(
123    address _recipient,
124    string memory _transactionNumber,
125    uint256 _amount,
126    address _srcToken,
127    address _destToken,
128    uint8 _destChainId
129  ) external nonReentrant {
130    bytes32 message =
131      keccak256(
132        abi.encodePacked(
133          _transactionNumber,
134          _amount,
135          _srcToken,
136          _destToken,
137          _recipient,
138          uint256(_destChainId)
139        )
140      );
141
142    require(
143      swaps[message].state == State.Empty,
144      "Bridge: swap is not empty state or duplicate secret"
145    );
146
147    require(
148      address(liquidityPools[_srcToken]) != address(0),
149      "Bridge: pool is not registered"
150    );
151
152    // lock the swap amount in the source pool
153    uint256 balanceBefore = IERC20(_srcToken).balanceOf(address(liquidityPools[_srcToken]));
154    IERC20(_srcToken).safeTransferFrom(
155      msg.sender,
156      address(liquidityPools[_srcToken]),
157      _amount
158    );
159    uint256 netAmount = IERC20(_srcToken).balanceOf(address(liquidityPools[_srcToken])) -
         balanceBefore;
```

```
160
161     swaps[message] = SwapData({
162       initTimestamp: block.timestamp,
163       initiator: msg.sender,
164       recipient: _recipient,
165       amount: netAmount,
166       srcToken: _srcToken,
167       destToken: _destToken,
168       destChainId: _destChainId,
169       state: State.Active
170     });
171
172     emit SwapInitialized(
173       swaps[message].initTimestamp,
174       msg.sender,
175       _recipient,
176       netAmount,
177       _srcToken,
178       _destToken,
179       _destChainId,
180       _transactionNumber
181     );
182   }
```

PontoonBridge.sol

```
83    function addLiquidity(uint256 _amount) external nonReentrant {
84        uint256 balanceBefore = IERC20(token).balanceOf(address(this));
85        IERC20(token).safeTransferFrom(msg.sender, address(this), _amount);
86        uint256 netAmount = IERC20(token).balanceOf(address(this)) - balanceBefore;
87
88        uint256 lpTokenAmount = netAmount * (10**factor);
89
90        liquidity[msg.sender].lpTokenBalance += lpTokenAmount;
91        liquidity[msg.sender].unlockTime = block.timestamp + lockPeriod;
92
93        _mint(msg.sender, lpTokenAmount);
94    }
```

PontoonPool.sol

Impact The attacker could swap for more `_destToken`. The attacker can also obtain much more LP tokens of PontoonPool by adding small amount of liquidity.

Suggestion Fix the reentrancy vulnerability in PontoonBridge and PontoonPool.

2.1.2 Unchecked `_destChainId`

Status Confirmed and fixed.

Description There is no validation of the `_destChainId` parameter of the `swap()` function in PontoonBridge.

Impact if `_destChainId` equals to current chain Id, a swap operation will be conducted in a same chain.

Suggestion A validation for `_destChainId` should be applied, in order to prevent swapping in a same chain.

2.1.3 Unchecked Parameter

Status Confirmed and fixed.

Description The `swap` function in PontoonBridge does not check the `_amount` parameter.

Impact If `_amount` == 0, the function will succeed (however, the transaction should be reverted if user swaps nothing).

Suggestion Check `_amount` before the actual swap logic.

2.1.4 Incorrect Implementation of `safeRewardTransfer()`

Status Confirmed and fixed.

Description The function `safeRewardTransfer()` in Vault has the `onlyOwner()` modifier. However, the vault's owner is the address which called `PontoonFarmFactory.deploy()`, and is not the PontoonFarm. This means any call from PontoonFarm to `Vault.safeRewardTransfer()` will fail, thus breaking the entire farming logic.

```
19  function safeRewardTransfer(
20    IERC20 rewardToken,
21    address to,
22    uint256 amount
23    ) public onlyOwner {
24      rewardToken.safeTransfer(to, amount);
25    }
```

Vault.sol

Impact The PontoonFarm can not call the function safeRewardTransfer() in Vault.

Suggestion Revise `safeRewardTransfer()` so that it can be accessed by PontoonFarm.

2.2 DeFi security

2.2.1 Neglection of Deflation & Inflation Tokens

Status Confirmed and fixed.

Description The contracts do not consider deflation or inflation tokens (in PontoonBridge, PontoonPool, PontoonFarm), or any token that has inconsistent values in the transfer function and actual transferred amounts. The contracts directly use the parameter `_amount` as token's actual transfer amount, which is not the case for deflation or inflation token.

Impact In PontoonFarm, the reward could be incorrectly.

Suggestion Revise the implementation so that it is compatible with deflation & inflation tokens, or any other tokens with inconsistent transfer values.

2.2.2 Neglection of Token with Non-Standard Decimals

Status Acknowledged.

Description The projects assume that the decimals of the underlying token are less than or equal to 18. However, some tokens may have larger decimals. In this case, the constructor will revert because of (built-in) overflow check. What's more, in some cases, token decimals cannot be retrieved (since it is an OPTIONAL feature of EIP-20). The project owner needs to manually review the tokens supported in the bridge and ensures that the decimals comply with the assumption.

Impact The constructor will revert because of (built-in) overflow check for tokens with a larger decimals.

Suggestion Any token that is going to be used by these contracts should be reviewed by the project owners to ensure the decimals comply with the assumption.

2.2.3 Unchecked Equivalence of Token Decimals

Status Acknowledged.

Description In PontoonFarm, there is no check to ensure the decimals of the staking token and all the reward tokens are equal. The requirement is mentioned in README.md, i.e. "uint256 rewardTokenDecimals: The staking token and all reward tokens should be ERC20 tokens with the same number of tokens".

```
50  constructor(
51    uint256 _startBlock,
52    uint256 blockDuration,
53    Vault _vault,
54    IERC20 _stakingToken,
55    IERC20[] memory _rewardTokens,
56    uint256[] memory _rewardPerBlock,
57    uint256 claimDelayDuration,
58    uint256 rewardTokenDecimals
59    ) {
60    require(_rewardTokens.length == _rewardPerBlock.length, "length mismatch");
61    require(blockDuration > claimDelayDuration, "claim cannot exceed blockDuration");
62    require(_rewardTokens.length <= 4, "cannot have more that 4 reward tokens");
63    require(isContract(address(_vault)), "vault address not valid");
64    require(isContract(address(_stakingToken)), "staking token address not valid");
65
66    for (uint256 i = 0; i < _rewardTokens.length; i++) {
67      require(isContract(address(_rewardTokens[i])), "rewardToken address not valid");
68      accRewardPerShare.push(0);
69    }
70
71    startBlock = _startBlock;
72    rewardPerBlock = _rewardPerBlock;
73    endBlock = _startBlock + blockDuration;
74    claimDelayBlock = _startBlock + claimDelayDuration;
75    lastRewardBlock = block.number > _startBlock ? block.number : _startBlock;
76
77    vault = _vault;
78    stakingToken = _stakingToken;
79    rewardTokens = _rewardTokens;
80    normalizationFactor = 10 ** rewardTokenDecimals;
```

```
81    }
```

PontoonFarm.sol

Impact If tokens have different decimals, PontoonFarm may execute incorrectly.

Suggestion Check token decimals in `constructor()`.

2.2.4 No Cancellation Mechanism

Status Acknowledged.

Description Bridge does not provide any cancellation mechanism for swaps. When users invoke the `swap()` function in the PontoonBridge contract, the funds are transferred to the underlying pool. PontoonBridge does not provide any cancellation mechanism for this operation.

Impact Users cannot withdraw their funds if they want to cancel the swap.

Suggestion Add cancellation mechanism.

2.2.5 No Pause Mechanism

Status Confirmed and fixed.

Description No pause mechanism is provided for PontoonBridge and PontoonPool contracts. Emergency pause is a good development practice so that in any emergency case, the project owner can temporarily pause the smart contracts.

Suggestion Add emergency pause mechanism.

2.2.6 Inconsistent Functions Behaviors

Status Acknowledged.

Description The behaviors of functions `updateLockPeriod()` and `updateLockPeriodForPool()` in PontoonBridge are inconsistent. The storage variable `lockPeriod` is only used in PontoonBridge for `addPool()` and `updatePool()`. The `updateLockPeriod()` function only updates the `lockPeriod` inside the PontoonBridge contract, without updating any pools (the new lockPeriod will only apply to newly added or updated pool); However, the `updateLockPeriodForPool()` actively updates the `lockPeriod` for pool by calling `updateLockPeriod()` on the specified pool.

```
50  /**
51  * @dev update the lockPeriod;
52  *
53  * Requirements
54  *
55  * - `_lockPeriod` period of lock remove liquidity.
56  */
57  function updateLockPeriod(uint256 _lockPeriod) public onlyOwner {
58    lockPeriod = _lockPeriod;
59  }
60
61  /**
62  * @dev update the lockPeriod for the pool;
```

```
63  *
64  * Requirements
65  *
66  * - `_token` token address.
67  * - `_lockPeriod` period of lock remove liquidity.
68  */
69  function updateLockPeriodForPool(address _token, uint256 _lockPeriod)
70    public
71    onlyOwner
72  {
73    PontoonPool liquidityPool = liquidityPools[_token];
74    PontoonPool(liquidityPool).updateLockPeriod(_lockPeriod);
75  }
```

PontoonBridge.sol

Impact The functionality of `updateLockPeriod` and `updateLockPeriodForPool` is inconsistent.

Suggestion Revise the implementation of `updateLockPeriod` and `updateLockPeriodForPool` to make them consistent.

2.2.7 Transfer Tokens without Invoking the `stake()` Function

Status Confirmed and fixed.

Description In PontoonFarm, there is a potential risk that users can transfer staking tokens to Farm without calling stake() function or stake rewarding token.

Impact In this case,

- Since the reward calculation is based on `totalStaked`, it may be miscalculated. Furthermore, it results in the left over rewarding tokens in the vault contract.
- The function `withdrawRemainingRewards()` can not serve its purpose after the `endBlock`, because the requirement `stakingToken.balanceOf(address(this))` == 0 can not be satisfied. Therefore the left over rewarding tokens can not be withdrawn.
- The mis-transfered staking token is lock in the vault contract, too.

Suggestion Revise the implementation so that the contracts could handle the incorrect transfers.

2.2.8 Lock Period Bypass

Status Acknowledged.

Description Users are allowed to stake after the claimDelayBlock in PontoonFarm.

Impact This may bypass the lock period.

2.3 Additional Recommendations

2.3.1 Single Validator

Status Acknowledged.

Description In the `redeem()` function of PontoonBridge, only one signature is used to complete a cross-chain operation. This single signature can have single-failure point issues.

```
201  function redeem(
202  address _recipient,
203  address _initiator,
204  string memory _transactionNumber,
205  uint256 _amount,
206  address _srcToken,
207  address _destToken,
208  uint8 _destChainId,
209  uint8 _v,
210  bytes32 _r,
211  bytes32 _s
212  ) external nonReentrant {
213    RedeemData memory data;
214
215    data.message = keccak256(
216    abi.encodePacked(
217    _transactionNumber,
218    _amount,
219    _srcToken,
220    _destToken,
221    _recipient,
222    uint256(_destChainId)
223    )
224    );
225
226    require(
227    swaps[data.message].state == State.Empty,
228    "Bridge: swap is not empty state or duplicate secret and hash"
229    );
230
231    data.signer = ECDSA.getSigner(data.message, _v, _r, _s);
232    require(
233    data.signer == validator,
234    "Bridge: validator address is invalid"
235    );
236    ......
```

PontoonBridge.sol

Impact It may subject to the single point failure (e.g. The private key is leaked).

Suggestion Use multi-signature scheme.

2.3.2 Better Function Return Value

Status Confirmed and fixed.

Description In PontoonFarm, it may be better if function `userData()` returns the staked amount of the user.

2.3.3 Misspelling in Comments

Status Confirmed and fixed.

Description There are some typos which should be fixed in comments in PontoonFarm.

- line 96: claculates -> calculates.
- line 97: eill -> will.