Moonpool
PoseidonSwap Project

This is a free audit report
Published on 24th March, 2021

Poseidon Security Analysis

By MoonPool

Overview Abstract

In this report, we consider the security of the Poseidon project. Our main task is to find and describe security issues in the smart contracts of the platform to the team.

Limitations and use of the report

Broadly speaking, the assessments can not uncover all vulnerabilities of the given smart contracts, thus, it is not guaranteed that the system is secured even if no vulnerabilities are found. The focus of the assessments was limited to only 3 given smart contracts, other contracts were excluded (including external libraries or third party codes).

Note: The report is not investment advice.
The team can publish this report to the community.

Summary

We have found **1** medium severity issue and **9** low severity issues.

The source codes were mainly influenced from well-known projects on Ethereum and Binance Smart Chain.

Recommendations

We recommend the team to fix all issues, as well as tests coverage to ensure the security of the contracts.

Assessment Overview

Scope of the audit

Source codes for the audit was initially taken from the commit hash *369a9b5a8b3d00a1ee406a870a56adff3223ee99* with the following **4** contracts:

1. **PoseidonToken.sol**
2. **PoseidonMasterChef.sol**
3. **PoseidonTokenLocker.sol**
4. **Timelock.sol**

Other source codes are out of scope for this report.

Note: After the first audit iteration, the team has fixed some reported issues.

The new audit report was taken from the latest commit at branch master .

System Overview

The system overview is based on the provided documentation that is available on their website here .

1. Poseidon Token :

- A standard token that allows the owner to mint more tokens to a given address with a given amount. A 200 tokens will be pre-minted after the contract has been deployed.

- The owner of the contract will be set to the Poseidon MasterChef, which allows only MasterChef to mint more token.

2. Poseidon Token Locker

- A contract to contain all locked Poseidon tokens and distribute linearly later on. The contract contains 2 variables: **startReleaseBlock** and **endReleaseBlock** to indicate when the token is unlocked and the period of the distribution.

- Anyone can call the **lock** function before the **startReleaseBlock** to lock their Poseidon tokens to the contract.

- After **startReleaseBlock** , anyone can call the **unlock** function to withdraw the claimable tokens from the contract. The locked token amount will be distributed linearly from the **startReleaseBlock** to the **endReleaseBlock** .

3. Poseidon MasterChef

- Allow stakers to deposit the supported tokens (single tokens or LP tokens) to the contract and earn Poseidon token as the reward. Each supported token/pool will have different configurations, include:
- **allocPoint** : how many allocation points that are assigned to this token/pool. The Poseidon reward will be distributed for stakers of this token/pool based on the **allocPoint** .

- **depositFeeBP** : How many percent (in basis points) of deposited tokens will go to the fee address. 1 is 0.01% and 10,000 is 100%. The **depositFeeBP** is at most **4%**

- **lockedPercentage** : How many percent (in basis points) of reward will be locked in the Poseidon Token Locker. 1 is 0.01% and 10,000 is 100%.

4. Timelock

A standard timelock contract that allows the owner to submit any transactions which are only executed after the **delay** time. The minimum delay time is set as 24 hours.

Findings

We have found 1 medium severity issue and 9 low severity issues.

PoseidonToken

[Low] Potentially high gas consumption for transfer, transferFrom, mint, etc

This code is influenced by Compound token contract code to allow taking a snapshot on each action (transfer, transferFrom, mint, etc), that will be used to calculate voting power for the governance.
If the team does not want to use the token for their Governance, we recommend removing the snapshot mechanism to reduce gas consumption.
Comment from the team : We will add governance features later on, so would be useful to keep the current logic as it is

[Partial-Fixed] [Low] Code Convention

1. Line 9: Function order is incorrect. Mint function should be declared below state variable declaration, function order should be: external -> public -> internal.
2. Error message for require is too long: Some requires are using too long error messages, consider to make them shorter.
3. Line 240: Should avoid to use inline assembly.

Some issues have been fixed and not present in the latest version of the code.

PoseidonTokenLocker

[Low] Change type of poseidon from address to *IBEP20*

The contract is declared **poseidon** as **address** type and later cast it to *IBEP20* for all usages (lines: 68, 69, 70, 97).
Should consider to change the type to *IBEP20* to avoid casting, thus, reducing gas consumption.

The team decided not to fix this issue for the simplicity of the code.

[Low] Use local variable to save gas consumption

Line 63: **canUnlockAmount**
Should use a local variable for **block.number** to save gas consumption.
Line 91: **unlock**
Should use a local variable for **msg.sender** to save gas consumption.

This issue has been fixed and not present in the latest version of the code.

[Partial-Fixed] [Low] Code Convention

1. Line 19: Function order is incorrect. Function order should be: external -> public -> internal. Also the write function should go before the read function.
2. Error message for require is too long: Some requires are using too long error messages, consider to make them shorter.
3. Lack of comments in the code: Should add more comments to the code, especially for each function to describe its functionality.

Some issues have been fixed and not present in the latest version of the code.

PoseidonMasterChef

[Fixed] [Medium] Can not unstake if the locker startReleaseBlock is up

In the **safePoseidonTransfer** function, it calls the locker to lock tokens for users. The withdraw function is calling this safe transfer function to transfer rewards to users, as well as locking tokens to the locker. In case the **block.number** > **startReleaseBlock**, the locker will revert, causing the withdraw function to revert as well. As a result, users will have to use the **emergencyWithdraw** function, as well as losing part of their rewards.

This issue has been fixed and not present in the latest version of the code

[Fixed] [Low] Only transfer token when amount > 0 to save gas consumption

In the **safePoseidonTransfer** function, it transfers tokens as reward to the user, as well as tokens to locker. It should check if the **claimableAmount** and **lockedAmount** are positive before making the call to save gas.
This issue has been fixed and not present in the latest version of the code

[Low] Compress PoolInfo struct data to save gas

According to the requirements:

- **allocPoint** is not greater than 10,000, which is fit in 14 bits.

- **lastRewardBlock** : is the block number, which should be easily fit in 32 bits.

- **accPosPerShare** : is the accumulated Poseidon token per share, times 10^12. It can be fit in a uint128.

Consider to use uint16 for **allocPoint** and uint32 for **l astRewardBlock** , uint128 for **accPosPerShare** so that (**allocPoint, lastRewardBlock, accPosPerShare, depositBP, lockPercentage)** should be fit in a single uint256, thus, reducing storage save and load.

The team decided not to fix this issue for the simplicity of the code.

[Low] Compress UseInfo struct data to save gas

UserInfo has only 2 attributes of uint256: **amount** and **rewardDebt.** Both values should be fit in uint128, thus, can compress the **amount** and **rewardDebt** to a single uint256.

The team decided not to fix this issue for the simplicity of the code.

[Low] Use interface instead of contract for **poseidon** and **locker**

Consider to use **poseidon** as IBEP20 and **locker** as IPoseidonTokenLocker.

The team decided not to fix this issue for the simplicity of the code.

[Partial-Fixed] [Low] Code Convention

1. Error message for require is too long: Some requires are using too long error messages, consider to make them shorter.

2. Lack of comments in the code: Should add more comments to the code, especially for each function to describe its functionality.

3. Line length is too long: Line length shouldn't be too long, length <= 100 should be good for reading (line: 112, 131).

Some issues have been fixed and not present in the latest version of the code.

Testing

The team has not provided any tests for given contracts. We recommend adding full tests coverage to ensure the logic works as expected and the security of the smart contracts.

Comment from the team : all source codes are influenced from some well-known projects on Ethereum and Binance Smart Chain. Some of them have been well-tested and well-audited.

Final comments

- Deposited Fee can be at most **4%**, thus, the owner can not change the deposited fee to more than that once the MasterChef has been deployed.
- No migration code, thus, the owner can not change the LP token.

Buy our team a coffee:
0xA0b91Ec92dBE8Af20fc9058bE61a2da09508255c

Request audit from our team