Pontoon Staking

Smart Contract Audit

coinspect

Pontoon

coinspect

Pontoon Staking Audit

Prepared for Apocentre • September 2021

v210913

1. Executive Summary

In **August 2021**, Apocentre engaged Coinspect to perform a source code review of Pontoon Farm. The objective of the project was to evaluate the security of the smart contracts.

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk
0	**1**	**1**
Fixed -	Fixed 100%	Fixed 100%

The medium risk issue is about not taking into account that some contracts implement the ERC-20 interface but might discount transfer fees, and has the consequence that **users might end up with their rewards locked** (see PON-1). The low risk issue is about **missing recovery mechanisms** for undistributed rewards (see PON-2).

All findings have been correctly fixed.

2. Assessment and Scope

The audit started on August 19 and was conducted on the `master` branch of the git repository at https://github.com/pontoonfi/pontoon-staking as of commit `85f8fa3a461b949b784de369dafbfc500046d0fb` of **September 1, 2021.**

```
commit 85f8fa3a461b949b784de369dafbfc500046d0fb (HEAD -> main, origin/main, origin/HEAD)
Author: ppoliani <pp..@..ail.com>
Date:   Wed Sep 1 10:18:11 2021 +0100

    chore(*): add comments
```

The scope of the audit was limited to the following Solidity source files, shown here with their sha256sum hash:

```
fd2fc37176bbfb9721c213ad63288cf0544a8647854043596589d0ec6ef4df71  PontoonFarm.sol
167d9f2a418eb6eca21fd7a13b075c93d2e8b8a8408293e89daeafe8c57f3558  PontoonFarmFactory.sol
9abc1c320519bd67c56ccd608fe15d2e9ebb1210ffa71591a43a54977ca4c4fd  Utils.sol
16c47a7b0fac8cac8ad9fdc878908b2f8be440931d4dbabd3fe2b088a8e34bf1  Vault.sol
```

The contracts are specified to be compiled with Solidity 0.8.6, being the latest Solidity version 0.8.7

There are 26 tests using the hardhat stack. All tests worked correctly out-of-the-box and Coinspect verified that all tests passed. Coverage tools reported about 80% of line coverage and 70% of branching coverage for `PontoomFarm.sol` which is the main file of the project. The untested parts are the last branch of the `releaseRewards` function and the `withdrawRemainingRewards` function. Both functions perform calls to ERC20 transfers. Coinspect recommends covering this code with additional tests. The code quality is good and the contracts are easy to understand.

`PontoomFarm` is a staking contract used for rewarding users that lock tokens on it. The creators must deploy a `PontoomFarm` and a `Vault` contract specifying the starting block, duration, staking token, reward tokens and reward amount. Then, the `Vault` contract must be transferred to the `Farm` and lock the reward tokens in the `Vault` contract. There is a `PontoomFarmFactory` contract that simplifies this process except for the locking of tokens.

Once the farming begins, the users must approve the `PontoomFarm` contract for using the staking contract and later calling the `stake()` function for receiving the rewards once the period finishes.

Users can recover the staked funds at any time. If it is before a specified block it will be at the cost of receiving no rewards and must be performed by calling the `emergencyUnstake()` function which can also be used in unexpected situations to recover the funds. After the established period is completed, users can call `unstake()` and receive the accumulated reward.

If users only want to collect rewards at some point during the farming, they can call the `stake()` function setting the `amount` parameter to 0.

Users must verify that the contract is funded before staking.

3. Summary of Findings

Id	Title	Total Risk	Fixed
PON-1	Unverified transfer amount may lock rewards	Medium	✔
PON-2	Missing recovery option for undistributed rewards	Low	✔

4. Detailed Findings

PON-1 Unverified transfer amount may lock rewards

Total Risk	Impact	Location
Medium	Medium	`contracts/PontoonFarm.sol`
Fixed	**Likelihood**	
✔	Medium	

Description

The staked amount may be less than specified in the `stake` function. This could happen if `ERC20` tokens that discount a fee for each transaction are used.

When a user calls the `stake` function, the value given in the parameter could differ from the actual value received by the `PontoomFarm` even on successful transactions, but the `stake` function takes into consideration the value sent as an argument to the call.

```
[contracts/PontoonFarm.sol]
function stake(uint256 amount) external nonReentrant {
...
    if (amount > 0) {
        stakingToken.safeTransferFrom(
                address(msg.sender),
                address(this),
                amount
        );

        user.amount = user.amount + amount;
      }
```

This would break an invariant that should be held by the contract: the sum of all amounts must be equal or lower than the `totalStaked` value (the amount of staked tokens held by the farm contract). Causing two problems:

- When rewarding: there will not be enough tokens to be rewarded due to an error on the reward per share estimated.
- When unstaking: there will not be enough funds for unstaking the total amount corresponding to each user.

Recommendation

Compute the actual amount transferred as the difference between the balances before and after the transfer. This computed amount should be used instead of the amount specified in the call to the `stake()` function.

Status

Fixed in commit c4eb161e4cfcaeaaf40e1feb4f4cf556d83064f8.

PON-2	Missing recovery option for undistributed rewards	
Total Risk Low	Impact Low	Location `contracts/PontoonFarm.sol`
Fixed ✔	Likelihood Low	

Description

Some funds might become unrecoverable when the transferred balance to the `farm` exceeds the claimable values. This might happen in multiple scenarios and the code to demonstrate the issue can be found in the first appendix of this document.

Recommendation

Add a recovery option available after the farming has ended.

Status

Fixed in commit `c08dbe7955e154a0789bab8ede7c25f4228d5a30`.

5. Appendix I: Unrecoverable funds test

```
const { network } = require("hardhat");

function toBase(n, dec = 18) {
  return ethers.BigNumber.from(n).mul(ethers.BigNumber.from(10).pow(dec))
}

describe("PontoonFarm: reward calculations", () => {
  let Alice, Bob
  let TokenFactory, PontoonFarmFactory, VaultFactory
  let stakingToken
  let reward
  let vault, farm
  let accounts
  let duration = 1000
  let noop = () => {} // Used during development for non staking
  let mintValue = toBase(100)


   let delayStake = async (account, farm, vault, staking, reward, step) =>
{
    if (step == duration / 2) {
      await farm.connect(account.signer).stake(mintValue)
    }
    if (step == duration) {
      await farm.connect(account.signer).unstake(mintValue)
    }
  }

   let betAllThenExit = async (account, farm, vault, staking, reward, step)
=> {
    if (step == 0) {
      await farm.connect(account.signer).stake(mintValue)
    }
    if (step + 1 == duration / 2) {
      await farm.connect(account.signer).emergencyUnstake()
    }
  }

  beforeEach(async () => {

    [Alice, Bob] = await ethers.getSigners()
    accounts = [
      {signer: Alice, name: "Alice", strategy: delayStake},
      {signer: Bob, name: "Bob", strategy: betAllThenExit}
    ]
    TokenFactory = await ethers.getContractFactory("BEP20Mock");
    PontoonFarmFactory = await ethers.getContractFactory("PontoonFarm");
    VaultFactory = await ethers.getContractFactory("Vault");

    vault = await VaultFactory.deploy()

    stakingToken = await TokenFactory.deploy()
    reward = await TokenFactory.deploy()
  })

  it('Delay bet and emergency exit', async() => {

    await network.provider.send("evm_setAutomine", [false]);
    const {number: blockNumber} = await ethers.provider.getBlock()

    await reward.mint(vault.address, mintValue)

    farm = await PontoonFarmFactory.deploy(
      blockNumber + 2,
      duration,
      vault.address,
      stakingToken.address,
      [reward.address],
      [mintValue.div(duration)],
      3 * duration / 4,
      18
    )

    await network.provider.send("evm_mine", [])
    await vault.transferOwnership(farm.address)
    for (let account of accounts) {
      await stakingToken.mint(account.signer.address, mintValue)
    }
    for (let account of accounts) {
          await  stakingToken.connect(account.signer).approve(farm.address,
mintValue)
    }
    await network.provider.send("evm_mine", [])


    for (let step = 0; step <= duration; step++) {
      for (let account of accounts) {
         await account.strategy(account, farm, vault, stakingToken, reward,
step)
      }
      await network.provider.send("evm_mine", [])
    }

    for (let account of accounts) {
      let balance = await reward.balanceOf(account.signer.address)
      let percentage = balance.mul(100).div(mintValue)
        console.log("Balance of %s is %s and about %%%s", account.name,
balance, percentage)
    }
  })
});
```

6. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.