Security Assessment

8Pay Staking Contracts

Jun 2nd, 2021

Table of Contents

Summary

This report has been prepared for 8Pay Staking smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	8Pay Staking Contracts
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/8pay/staking-contracts
Commits	f1f8c14bda9e03cc766398c2022c1e7a736f3e25 84c23874107c746059d2bee61d8e19899e4f3ab3 4ee872410b97150845524c632e43ab877b1c7b7b 0e12551c537e71f55f88881409c3b92d09e2de39 6e0eb56ab1e23294291e68f6b6556ffd38f8f991 465e3fe7f779155706dffb614c61f1dfa9c179e6 c40b0f66e2ec7ec856cc7388dcc16f2f199110c4 f0112615ede4f97b7731af9c828edc7aa781a646 c6f2dc9f8bbd0c4fa95651e0b335eb783b29c4e5 5c2c27437d0bf71fc92525c83286d06fd32d2951 fbfc217bd3c0546648912963faad3ddf33a5a0c8 64c863c0414a9772dd7d813c66c2c651667767ec bd2ecb09c9e623bafa134cb2d1c225467b9b9919 2888683387c0b54cde05a14ca5d6f5fe1d054b8f

Audit Summary

Delivery Date	Jun 02, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

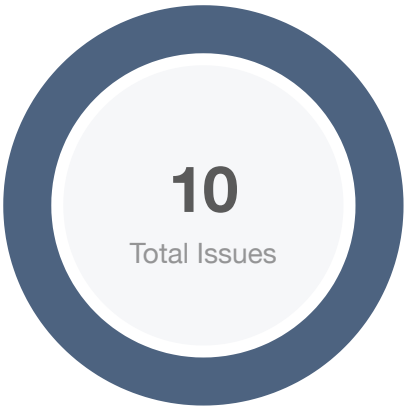
Vulnerability Summary

Total Issues	10
Critical	0
Major	0
Medium	0
Minor	0
Informational	10
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
RTY	RewardTreasury.sol	3391d910291dadf6f24635f2c30052b408723e5d6e780f57916c4df958acc17d
SPY	StakePool.sol	32add6674a1643e03cf482406a7361852ebaae5891a620ecae87576354bbecf7

Findings



Critical	**0** (0.00%)
Major	**0** (0.00%)
Medium	**0** (0.00%)
Minor	**0** (0.00%)
Informational	**10** (100.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
RTY-01	Potential loss of reward token funds through `sendReward`	Control Flow	Informational	Resolved
SPY-01	Potential loss of reward token funds through `deposit`	Data Flow	Informational	Resolved
SPY-02	Potential loss of staking and reward token funds through `withdraw`	Data Flow	Informational	Resolved
SPY-03	Non-conforming `claimReward` function can lead to loss of reward token funds	Data Flow	Informational	Resolved
SPY-04	Potential loss of staking token funds through `emergencyWithdraw`	Data Flow	Informational	Resolved
SPY-05	Redundant user reward amount requirement	Gas Optimization	Informational	Resolved
SPY-06	Redundant pool update	Gas Optimization	Informational	Resolved
SPY-07	Owner can claim arbitrary reward amount	**Centralization / Privilege**	**Informational**	**Acknowledged**
SPY-08	Owner can set reward per block repeatedly	**Centralization / Privilege**	**Informational**	**Acknowledged**
SPY-09	Owner can set staking end block repeatedly	**Centralization / Privilege**	**Informational**	**Acknowledged**

RTY-01 | Potential loss of reward token funds through `sendReward`

Category	Severity	Location	Status
Control Flow	Informational	RewardTreasury.sol: 28~30	Resolved

Description

The external `sendReward` function in the `RewardTreasury` contract is called often from within the `StakePool` contract. Due to this implementation, any re-entering calls in the `StakePool` contract which also call the `sendReward` function can allow draining of the reward token funds.

Recommendation

Since the project already depends on the `@openzeppelin/contracts` node module, consider importing and inheriting from the `ReentrancyGuard` contract and adding the `nonReentrant` modifier to the external `sendReward` function in order to prevent re-entering calls from draining the funds of the reward token:

```
import { ReentrancyGuard } from "@openzeppelin/contracts/security/ReentrancyGuard.sol";
```

```
contract RewardTreasury is Ownable, ReentrancyGuard {
```

```
function sendReward(address to, uint256 amount) external onlyOwner nonReentrant {
```

Alleviation

The recommendation was not taken into account, but has been deemed to be resolved as of commit `4ee872410b97150845524c632e43ab877b1c7b7b` due to the RewardTreasury instance being owned and managed by the StakePool instance.

SPY-01 | Potential loss of reward token funds through `deposit`

Category	Severity	Location	Status
Data Flow	Informational	StakePool.sol: 164~168	Resolved

Description

The external `deposit` function in the `StakePool` contract does not contain any form of access restriction and ignores the check effects interactions patterns due to not updating the user's staked amount until after the token transfer on line 168, which allows re-entering calls to drain the funds of the reward token through repeat calls to the internal `_sendReward` function if the caller has a non-zero balance.

Recommendation

Since the project already depends on the `@openzeppelin/contracts` node module, consider importing and inheriting from the `ReentrancyGuard` contract and adding the `nonReentrant` modifier to the external `deposit` function in order to prevent re-entering calls from draining the funds of the reward token:

```
import { ReentrancyGuard } from "@openzeppelin/contracts/security/ReentrancyGuard.sol";
```

```
contract StakePool is Ownable, ReentrancyGuard {
```

```
function deposit(uint256 amount) external nonReentrant {
```

Alleviation

The recommendation was indirectly taken into account by refactoring the order of statements in order to prevent re-entrancy from being possible in commit `4ee872410b97150845524c632e43ab877b1c7b7b`.

SPY-02 | Potential loss of staking and reward token funds through `withdraw`

Category	Severity	Location	Status
Data Flow	Informational	StakePool.sol: 191~197	Resolved

Description

The external `withdraw` function in the `StakePool` contract does not contain any form of access restriction and ignores the check effects interactions patterns due to not updating the user's staked amount until after the token transfer on line 193, which allows re-entering calls to drain the funds of both the reward token through repeat calls to the internal `_sendReward` function and the staking token upon re-entering the transfer call.

Recommendation

Consider moving the transfer on line 193 to the end of the function in order to properly follow the check effects interactions pattern. Consider also importing and inheriting from the `ReentrancyGuard` contract and adding the `nonReentrant` modifier to the external `withdraw` function in order to prevent re-entering calls from draining the funds of the reward and staking tokens:

```
import { ReentrancyGuard } from "@openzeppelin/contracts/security/ReentrancyGuard.sol";
```

```
contract StakePool is Ownable, ReentrancyGuard {
```

```
function withdraw(uint256 amount) external nonReentrant {
    require(amount > 0, "Pool: withdraw amount is zero");
    UserInfo storage userInfo = usersInfo[msg.sender];
    require(userInfo.amount >= amount, "Pool: nothing to withdraw");

    _updatePool();
    _sendReward(msg.sender);

    userInfo.amount -= amount;
    userInfo.rewardDebt = userInfo.amount * accRewardPerShare / 1e12;
    totalStakedTokens -= amount;

    emit Withdraw(msg.sender, amount);
    stakeToken.safeTransfer(msg.sender, amount);
}
```

Alleviation

The recommendation was taken into account by refactoring the order of statements in order to prevent re-entrancy from being possible in commit `4ee872410b97150845524c632e43ab877b1c7b7b`.

SPY-03 | Non-conforming `claimReward` function can lead to loss of reward token funds

Category	Severity	Location	Status
Data Flow	Informational	StakePool.sol: 205~207	Resolved

Description

The implementation of the external `claimReward` function in the `StakePool` contract does not conform to the design of the other external functions within the `StakePool` contract due to failing to verify if the calling user's reward amount is greater than zero and failing to update the pool prior to calling the internal `_sendReward` function. The function also does not contain any form of access restriction and allows re-entering calls to drain the funds of the rewards token by making repeated calls to the internal `_sendReward` function.

Recommendation

Consider adding a requirement to the external `claimReward` function in order to verify that the calling user's reward amount is greater than zero, followed by making a call to the internal `_updatePool` function. Since the project already depends on the `@openzeppelin/contracts` node module, consider also importing and inheriting from the `ReentrancyGuard` contract and adding the `nonReentrant` modifier to the external `claimReward` function in order to prevent re-entering calls from draining the funds of the reward token:

```
import { ReentrancyGuard } from "@openzeppelin/contracts/security/ReentrancyGuard.sol";
```

```
contract StakePool is Ownable, ReentrancyGuard {
```

```
function claimReward() external nonReentrant {
    UserInfo storage userInfo = usersInfo[user];
    require(userInfo.amount > 0, "Pool: nothing to claim");
    _updatePool();
    _sendReward(msg.sender);
}
```

Alleviation

The recommendation was found to be indirectly taken into account as of commit `4ee872410b97150845524c632e43ab877b1c7b7b` by refactoring the function to follow the implementations within the contract.

SPY-04 | Potential loss of staking token funds through `emergencyWithdraw`

Category	Severity	Location	Status
Data Flow	Informational	StakePool.sol: 217~224	Resolved

Description

The external `emergencyWithdraw` function in the `StakePool` contract does not contain any form of access restriction and ignores the check effects interactions patterns due to not updating the user's staked amount until after the token transfer on line 214, which allows re-entering calls to drain the funds of the staking token.

Recommendation

Consider moving the transfer on line 214 to the end of the function in order to follow the check effects interactions pattern. Since the project already depends on the `@openzeppelin/contracts` node module, consider importing and inheriting from the `ReentrancyGuard` contract and adding the `nonReentrant` modifier to the external `emergencyWithdraw` function in order to prevent re-entering calls from draining the funds of the staking token:

```
import { ReentrancyGuard } from "@openzeppelin/contracts/security/ReentrancyGuard.sol";
```

```
contract StakePool is Ownable, ReentrancyGuard {
```

```
function emergencyWithdraw() external nonReentrant {
    UserInfo storage userInfo = usersInfo[msg.sender];

    require(userInfo.amount > 0, "Pool: nothing to withdraw");

    uint256 amount = userInfo.amount;
    emit EmergencyWithdraw(msg.sender, amount);

    userInfo.amount = 0;
    userInfo.rewardDebt = 0;

    stakeToken.safeTransfer(msg.sender, amount);
}
```

Alleviation

The recommendation was indirectly taken into account by refactoring the order of statements in order to prevent re-entrancy from being possible in commit `4ee872410b97150845524c632e43ab877b1c7b7b`.

SPY-05 | Redundant user reward amount requirement

Category	Severity	Location	Status
Gas Optimization	Informational	StakePool.sol: 272	Resolved

Description

The internal `_sendReward` function in the `StakePool` contract contains a requirement that the supplied user's reward amount should be greater than zero, which is redundant due to the requirements being verified prior to calling the `_sendReward` function.

Recommendation

Consider removing the requirement that the supplied user's reward amount should be greater than zero on line 272 in order to save on the overall cost of gas.

Alleviation

The recommendation was found to be taken into account in commit `84c23874107c746059d2bee61d8e19899e4f3ab3`.

SPY-06 | Redundant pool update

Category	Severity	Location	Status
Gas Optimization	Informational	StakePool.sol: 274	Resolved

Description

The internal `_sendReward` function in the `StakePool` contract makes a call to the internal `_updatePool` function, which is redundant due to the pool already having already been updated prior to calling the `_sendReward` function.

Recommendation

Consider removing the call to the internal `_updatePool` function on line 274 in order to save on the overall cost of gas.

Alleviation

The recommendation was found to be taken into account in commit `84c23874107c746059d2bee61d8e19899e4f3ab3`.

SPY-07 | Owner can claim arbitrary reward amount

Category	Severity	Location	Status
Centralization / Privilege	**Informational**	StakePool.sol: 115~117	**Acknowledged**

Description

The external `removeRewardTokens` function in the `StakePool` contract allows the owner of the pool to send themselves an arbitrary amount of reward tokens.

SPY-08 | Owner can set reward per block repeatedly

Category	Severity	Location	Status
Centralization / Privilege	**Informational**	StakePool.sol: 126~130	**Acknowledged**

Description

The external `setRewardPerBlock` function in the `StakePool` contract allows the contract owner to change the amount of reward tokens distributed per block, which requires trust in the owner to not modify at the last minute.

SPY-09 | Owner can set staking end block repeatedly

Category	Severity	Location	Status
Centralization / Privilege	**Informational**	StakePool.sol: 139~149	**Acknowledged**

Description

The external `setEndBlock` function in the `StakePool` contract allows the owner of the contract to set the staking end block multiple times, which requires trust to remain consistent.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.