IQ NFT Staking Security Analysis

by Pessimistic

This report is public

July 4, 2024

Abstract

In this report, we consider the security of smart contracts of IQ NFT Staking project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of IQ NFT Staking smart contracts. We described the audit process in the section below.

The initial audit showed one critical issue: The stakers are not able to claim their rewards. The audit also revealed two issues of medium severity: Unchecked returned value, Project roles. Moreover, several low-severity issues were found.

After the audit, the developers provided a new version of the code that includes fixes for most of the findings and provides comments on the M02 issue.

After that, the developers provided us with a new code version. In this update, they introduced new functionality, changing the code significantly. The new version of the code contained C02 critical issue, M03 and M04 medium severity issues, several low severity issues and some notes.

Afterwards, the developers updated the codebase. In this update, they fixed all the issues found during the review of the previous code update. Moreover, M03 and N03 issues were removed from the report as false positives. A new M05 medium severity issue was found during the review and was fixed by the developers later.

The overall code quality of the project is good.

General recommendations

We do not have any recommendations for the project.

Project overview

Project description

For the audit, we were provided with IQ NFT Staking project on a private GitHub repository, commit 8220867e2b07df7e850f9016915a8c9088eefa90.

The scope of the audit included:

- **contracts/staking/IIQNftStaking.sol**;
- **contracts/staking/IQNftStaking.sol**.

The documentation for the project included **IQNftStaking documentation.pdf** (sha1sum 7c03589173d63a6ef434a4152589bf9303c10fa6).

All 27 tests pass successfully. The code coverage is 77.08%.

The total LOC of audited sources is 245.

Codebase update #1

After the initial audit, the codebase was updated. For the recheck, we were provided with commit d79b2d9c27bba01491e7acaace1d7dc3e678c399. This update included the fixes for all issues except for the M02 issue, comments on the M02 issue. In addition to this, the developers implemented additional tests.

All 32 tests pass successfully. The code coverage increased to 100%.

Codebase update #2

After the previous codebase update, the developers provided us with a new version of the code, commit bb5f411f71372847677bd32b762f694d908c6289. The update contained new functionality and significant protocol changes. Several new issues were found within the updated codebase: C02 critical issue, M03 and M04 medium severity issues and several low and notes severity issues. M02 issue was updated according to the new functionality.

The scope of the audit included:

- **contracts/staking/IIQNftStaking.sol**;
- **contracts/staking/IQNftStaking.sol**;
- **contracts/staking/IStakingManager.sol**;
- **contracts/staking/StakingManager.sol**.

All 55 tests pass successfully. The code coverage increased to 100%.

Codebase update #3

After the latest code update, we received the updated version for the recheck with commit 37ba330cb40b15a9b691e0fa46dc763dbda2c244. This update contained fixes for all the issues from the previous review and introduced a new M05 issue, which was fixed with commit 142d0b804841438fcc1f9712e547839b4584818d. M03 and N03 issues were removed from the report as false positives.

The number of tests increased to 61, and all of them passed successfully. The code coverage is 100%.

Audit process

We started the audit on April 12, 2024 and finished on April 15, 2024.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project. After a discussion, we performed preliminary research and started the review.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Users should be able to unstake their NFTs at any point in time;
- Users should be able to claim their reward tokens (see M02);
- NFT tokens cannot be withdrawn by the malicious actor;
- The signatures should not be replayable across the stakers;
- An owner should have access to the remaining amount of the reward tokens;
- Compliance of the code with the provided documentation.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts. We also sent the results to the developers in the text file.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

After the initial audit, we discussed the results with the developers. On April 24, 2024, the developers provided us with an updated version of the code. This update fixed most of the issues highlighted in our report and increased the code coverage.

We manually reviewed the updated codebase and also scanned it with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules.

After that, the report was updated.

Later, developers updated the codebase. We started the audit of the new code on June 25, 2024, and finished on June 28, 2024. Developers changed the logic of the **IQNftStaking** contract and added a new **StakingManager** contract to the scope. Before the review, we contacted the developers to understand the new functionality of the project better. Several new issues were found during the review.

Furthermore, we rescanned the updated codebase with tools mentioned above: static analyzer Slither with our plugin Slitherin, and Semgrep rules for smart contracts. We shared the text output of the Semgrep tool with the developers.

After that, we updated the report.

Following the previous code review, we discussed the results with the developers. On July 4, 2024, the developers provided us with an updated version of the code. This updated resolved all the issues found during the previous audit. A new M05 issue was added to the report and was fixed later by the developers. M03 and N03 issues were marked as false positives and were removed from the final report.

We preformed manual review of the updated codebase and also rescanned it with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules.

Finally, the report was updated.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

C01. The stakers are not able to claim their rewards (fixed)

After staking their NFTs, users should have an ability to claim the rewards by invoking the `claimTokens` function. However, the function accounts the claimed tokens internally without transferring them. In case the transfers are about to be handled by the `_proofSource`, an approval should be given to the backend.

Note that the reward tokens are not stuck, since the owner of the staking pool can deactivate the staking and withdraw all the reward tokens.

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

C02. No ownership transfer (fixed)

When the new staking contract is created with `StakingManager.deployNftStaking` function, the ownership is not transferred to the intended user address at line 97 of the **StakingManager** contract. It breaks important functionality as rewards cannot be deposited into the staking contract. Consider transferring the ownership to the deployer and covering this scenario with additional tests.

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

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Unchecked returned value (fixed)

Currently, any **ERC20** token could be accepted as a `_rewardToken` in order to reward the NFT stakers. Some older **ERC20** token versions either return or do not return a boolean value instead of revert upon transferring the tokens. An example of such token could be **USDT**. Consider utilizing the **SafeERC20** library from OpenZeppelin.

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

M02. Project roles (commented)

In the current design, an owner has an ability to deactivate the staking and withdraw all the reward tokens from the pool at any time. Moreover, the functionality behind the `stake`, `claimTokens` and `withdraw` functions heavily relies on the signatures from the backend.

The owner of the **StakingManager** contract can withdraw accumulated fees, change deployment price and determine fees, and change staking manager for staking contracts.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from the developers:*
>
> *Pool Withdrawal Restrictions: The staking owner can only withdraw unallocated tokens from the pool. Tokens accrued as rewards by users are protected and cannot be accessed by the owner, ensuring security for stakers.*
>
> *User Protection: Users maintain full control to withdraw their NFTs and corresponding rewards at any time, even if staking is deactivated.*
>
> *Secure Signatures: We use EIP712 for secure, structured signatures, with secret key management handled by Google's Secrets Manager. This setup ensures high security.*
>
> *Simplicity and Efficiency: Adding multisig would complicate our EIP712 setup, requiring additional variables and reducing the efficiency of transactions. Our current design balances security and efficiency.*

M04. Possible reentrancy (fixed)

`IQNftStaking.claimTokens` does not contain `nonReentrant` modifier as other external functions. Moreover, it contains an external call at line 228 of the **IQNftStaking** contract to possibly a malicious token contract. Additionally, several storage variables are updated after the external call at lines 231-234.

Mallicious user can obtain two signatures for tokens claiming and supply them both within one transaction through reentrancy, ignoring `_claimDelay` feature.

Consider following the CEI (checks-effects-interactions) pattern and adding `nonReentrant` modifier to the `claimTokens` function.

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

M05. Possible unrestricted staking deployment (fixed)

The signature for staking deployment at `StakingManager.deployNftStaking` does not include `msg.sender`, allowing deployment of staking contract with any valid signature and a corresponding `nonce` value of a deployer. Consider including the contract owner to the deployment signature.

The issue has been fixed with commit 142d0b804841438fcc1f9712e547839b4584818d.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Incorrect calculation of tokens withdrawn by owner (fixed)

After deactivating the staking, an owner is able to withdraw some of the tokens used to reward the stakers that remain in the contract. Since the owner can withdraw the tokens in multiple separate transactions, the variable `_tokensWithdrawedByOwner` may contain an incorrect amount of tokens being withdrawn.

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

L02. Unexpected revert in totalTokensLeft (fixed)

Currently, `totalTokensLeft` is calculated based on the `_poolSize`, `_totalTokensClaimed`, `_tokensWithdrawedByOwner` variables. However, the following invariant, which is required to be hold:

```
_totalTokensClaimed + _tokensWithdrawedByOwner <= _poolSize
```

might be broken, since the contract does not transfer the tokens upon claiming, but instead, accounts them internally. This causes an unexpected revert upon invoking `totalTokensLeft` function.

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

L03. Possible gas optimization (fixed)

In the current design, the `withdraw` function calls `_removeNftFromStaking` for each NFT that is about to be withdrawn, which itself, in the worst case scenario, iterates over the whole `_stakedTokens[user]` array. It might be problematic in case of a large number of tokens being withdrawn.

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

L04. Deployment fee (fixed)

Any leftovers from ether sent during the `deployNftStaking` call are not returned to the deployer, and the check at line 84 of the **StakingManager** contract allows these leftovers. Consider checking that the message value is exactly equal to `_deploymentPrice` during deployment.

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

L05. Incorrect check for amount (fixed)

The amount to withdraw is checked incorrectly at line 133 of the **StakingManager** contract as it has `uint256` type and cannot be less than zero. Consider changing the check to `amount == 0`.

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

L06. Unnecessary unchecked blocks (fixed)

There are several places in **IQNftStaking** contract where `unchecked` blocks do not have any effects. Consider removing `unchecked` statements at lines 249 and 272, as there are no arithmetic operations impacted by these blocks.

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

L07. _totalRewardAccrued is not used (fixed)

The value of `_totalRewardAccrued` state variable of the **IQNftStaking** contract is set in `deactivateStaking` function, but the variable is never used and cannot be accessed externally as it is declared as private.

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

L08. Setters do not emit events (fixed)

There are several setters in the **StakingManger** contract that do not emit any events:

- `StakingManager.setDeploymentPrice`;
- `StakingManager.setIndividualContractBatchTransactionFee`;
- `StakingManager.setStakingManager`;
- `StakingManager.setBatchTransactionFee`.

Consider emitting events in setter functions to inform other users about critical parameter changes.

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

L09. Immutable (fixed)

Some variables are written only in constructors and can be declared `immutable` to reduce gas consumptions:

- `_proofSource` in the **IQNftStaking** contract;
- `_nftCollection` in the **IQNftStaking** contract;
- `_proofSource` in the **StakingManager** contract.

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

L10. Public vs external (fixed)

Consider converting functions that are not used internally from public to external.

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

Notes

N01. No initial claim delay (fixed)

The value of `_lastClaimedTimestamp` of the **IQNftStaking** contract is equal to zero for a new staker before the first rewards claiming. It allows one to deposit NFT, claim rewards (without delay), withdraw NFT, and repeat the cycle from another address if no additional checks are performed on a backend that is responsible for claiming signatures.

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

N02. Possible to claim for another user (fixed)

Anyone can call `IQNftStaking.claimTokens` function to collect staking rewards for a particular `staker`. It is possible to call this function every `_claimDeplay` period. It might not be the most efficient strategy for the staker, e.g., if there is a planned deployment of the AMM pool with the reward tokens in `_claimDelay / 2` seconds, it is possible to call `claimTokens` for someone now and lock claiming for the next `_claimDelay` period, leaving some rewards unavailable when the pool is launched.

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

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Oleg Bobrov, Security Engineer
Rasul Yunusov, Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

July 4, 2024