IQ Staking Security Analysis

by Pessimistic

This report is public

December 6, 2023

Abstract

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

The audit showed one critical issue: Zero rewards for several current plans. The initial audit also revealed several issues of medium severity: Bug in the code, No documentation, No check for stake amount bounds, No guarantee for the reward, Insufficient test coverage, Problems with deleting plans, Possible loss of rewards, Possible plan overwriting and Owner role. Moreover, several low-severity issues were found.

After the initial audit, the developers updated the codebase. In this update, they fixed found critical issue, along with several medium and low-severity issues. However, this update also introduced several new issues.

After that, the developers provided us with a new version of the code. In this update, they fixed all medium and most of the low-severity issues.

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 Staking project on a public GitHub repository, commit e9908cf8aca0f3c3287637cc5fb011989a6655e0.

The scope of the audit included the whole repository.

There was no documentation provided for the audit.

All 19 tests pass successfully. The code coverage is 27.74%.

The total LOC of audited sources is 488.

Codebase update #1

After the initial audit, the developers provided us with a new version of the code, commit 68421f085a0500401417eb789bf7118a21191340, along with general documentation. The update includes a significant overhaul to the **Staking** contract and several changes to others. Additionally, this update introduces new issues of medium and low severity.

All 151 tests pass successfully. The code coverage is 97.22%.

Codebase update #2

After the previous codebase update, the developers provided us with a new version of the code, commit dccb5ff456ed0a4067da580f26bcf5c6bb12e20c. The update does not contain any new functionality and fixes most of the remaining issues from the report. With that update, the developers provided more comprehensive documentation. Based on it, we decided to mark M02 issue as fixed.

All 161 tests pass successfully. The code coverage is 94.31%.

Audit process

We started the audit on November 8, 2023 and finished on November 13, 2023.

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 specified those parts of the code and logic that require additional attention during an audit:

- The overall correctness of the staking and timelock logic;
- The coherency of the expected business logic and the code;
- The safety of user funds and their ability to receive all rewards and staked tokens.

In addition, the developers asked us to note code optimizations that are easy to implement and do not require complex code modifications.

We manually analyzed all the contracts within the scope of the audit and checked their logic.

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. Also, we sent the results to the developers in the text file.

We ran tests implemented later 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 November 15, 2023, the developers provided us with an updated version of the code, fixing some of the issues identified in our report and making significant changes to all parts of the project. Additionally, the developers thoroughly covered the code with tests and added general documentation for the project.

We ran the static analyzer Slither with our plugin Slitherin and verified their outputs.

We reviewed the fixes and the updated codebase.

Following the report, the developers made updates to the codebase. As a result, they increased the quality of the code and fixed most of the issues. We reviewed all the fixes, as well as the new documentation, marked issues with a "fixed" tag where necessary, and included comments from the developers.

In addition to this, we ran the static analyzer Slither with our plugin Slitherin and verified their outputs.

After that we updated the report.

Manual analysis

The contracts were completely manually analyzed, and 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. Zero rewards for several current plans (fixed)

Rewards are calculated based on a fraction of a year. According to the developers, the intended durations for staking plans are 1, 3, 6, and 12 months. For plans with durations less than 12 months, rewards will be zero. Additionally, for plans extending beyond 12 months, the division will be rounded down to the nearest integer.

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 potential problems related to incorrect system management. We highly recommend addressing them.

M01. Bug in the code (fixed)

In the **StakingManagement** contract there is an issue with the logical condition in the checks of the APY value at lines 58 and 74. Currently, the condition is `apy == 0 && apy > Constants.MAX_APY`, which will always return `false` because APY cannot be both equal to `0` and greater than `Constants.MAX_APY` at the same time. To address this, consider changing the logical operator from `&&` to `||`.

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

M02. No documentation (fixed)

The codebase is covered with NatSpec comments. However, they provide only a brief description of functionality and do not explain the overall project structure.

Proper documentation should explicitly explain the purpose and behavior of the contracts, their interactions, and the main design choices. We recommend adding it since it helps to find discrepancies between the business logic and the implementation.

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

M03. No check for stake amount bounds (fixed)

The **StakingManagement** contract includes several functions for setting limits on the minimum and maximum amounts for stake deposits. However, these checks are not used, which could result in unexpected stake amounts inside the **Staking** contract. It is important to ensure that the intended limits are enforced, and the checks are incorporated into the stake deposit process.

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

M04. No guarantee for the reward (addressed)

The owner must provide liquidity for the rewards timely. Currently, there is no code guaranteeing that all depositors will receive their rewards. Furthermore, in cases when there are insufficient rewards, the first withdrawers will take tokens from other stakers' deposits as rewards, potentially leaving some depositors without their stakes. As a result, these depositors will not be able to withdraw their funds until all rewards are sent to the contract.

The developers added a check that the balance of the reward pool is sufficient for the staking. However, we cannot verify that the tokens cannot be later withdrawn from the mentioned pool.

> *Comment from the developers: The design was built in a such way to **Timelock** contract would be created once and all time lock receivers would be defined immediately after deploy, excluding only future employees, but the supply of the tokens that we are going to have is fixed and will be defined up-front. Once the time lock is created we're going to provide allowance from vesting pool wallet to **Timelock** contract.*

M05. Insufficient test coverage (fixed)

In the audited commit, there were no tests for the **Staking** and **StakingManagement** contracts, only for the **BatchTimelock**. The code coverage was 27.74%. We notified the developers about that and later they implemented tests in the commit `47d3c17055f1dcd47e158ae1dd58566fc8e946d9` on the `staking-tests` branch. We ran tests on that version of the code and the code coverage became 74.04%. We still do not consider that coverage as a sufficient one, hence we recommend improving it.

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

M06. Problems with deleting plans (fixed)

The **StakingManagement** contract includes logic for deleting plans. However, if a plan is deleted while still used by some users, the stakes associated with that plan will not be withdrawable. It happens because the check inside `StakingManagement.getStakingPlan` function will revert due to the absence of the deleted plan.

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

M07. Possible loss of rewards (fixed)

The current implementation of the **BatchTimelock** contract allows a terminated account to withdraw its vested rewards until the `lock.terminationFrom` time. However, the claim operation will revert if the user attempts to claim rewards after the `terminationFrom` time. To address this, consider allowing the collection of rewards vested before termination even after the `terminationFrom` time.

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

M08. Possible plan overwriting (fixed)

The plans are identified by their IDs obtained using the length of `_stakingPlanIds`. However, if a plan is removed, adding a new plan after removal will overwrite the latest added plan. This behavior can alter the duration and APY of the plan, impacting stakes inside the **Staking** contract that used this `planId` before. Additionally, updating a plan using `updateStakingPlan` is not desired because it changes the parameters of existing stakes.

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

M09. Owner role

The `owner` role is crucial for the correct project operation. It has the following powers:

- Adding, updating, and deleting staking plans. This may alter the amount of the rewards for the users or lead to stuck funds (see M06 issue);
- Allowing and disallowing early stake withdrawals;
- Changing minimum and maximum stakes;
- The owners of the project are responsible for adding sufficient funds to the staking contract;
- Terminating and determinating token timelocks;
- Adding timelocks.

Some scenarios can lead to undesirable consequences for the project and its users, e.g., if the owner's private key becomes compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

In the Codebase update, the developers modified the role system. The list of changes is the following:

- The developers opted to use the **AccessControl** library instead of **Ownable**;
- The developers removed the ability to change staking plan parameters;
- In the **StakingManagement** contract, there is an option to change the **Staking** contract address.

M10. Terminated account receive all rewards immediately (fixed)

According to lines 178-181 of the **BatchTimelock** contract, the terminated account receive all their rewards before the `lock.terminationFrom`.

The issue has been fixed in commit 6b73be6d8f37d2d63eac117ce1b095cceebda84a.

M11. Incorrect reward pool size calculation (fixed)

In the **Staking** contract, the number of tokens users will receive for staking is tracked in the `_stakingPoolSize` variable. When a user deposits funds, the contract checks whether the balance of the `_stakingPool` address is sufficient. However, when a user withdraws funds, the value of `_stakingPoolSize` is not decreased, despite the fact that the `_stakingPool` address is required to hold fewer funds.

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

Low severity issues

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

L01. Dubious typecast (fixed)

In the **Staking.sol** file in `calculateStakeEarnings` function, `earningsPercentage` variable has an `uint16` type. Note that this variable will overflow in case when the user earns more than 655.36% of his total stake. Considering the fact that the maximum APY is 100%, this is an unlikely event, but we still recommend reviewing the necessity of the `uint16` type.

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

L02. Function visibility (fixed)

In the **BatchTimelock** contract, consider changing function visibility from `public` to `external` where possible in order to increase code readability.

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

L03. Gas consumption

In the **Staking** contract in `withdraw` function, it appears that the `onlyStakeOwner(stakeId)` check encapsulates the check for the existence of the stake. In this case, the `onlyExistingStake(stakeId)` check can be safely removed, as it is redundant. Additionally, `onlyExistingStake(stakeId)` check is performed later inside `calculateStakeEarnings`, resulting in a double validation of the same condition.

L04. Mark variables as immutable (fixed)

In the **Staking** contract, consider marking `_stakingToken` and `_stakingManagement` variables as `immutable` as they are not changed outside of the constructor.

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

L05. Mark variables as immutable (fixed)

In the **StakingManagement** contract, consider marking `stakingToken` variable as `immutable` as it is not changed outside of the constructor.

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

L06. Mark variables as immutable (fixed)

In the **BatchTimelock** contract consider marking `_token` and `_vestingPool` variables as `immutable` as they are not changed outside of the constructor.

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

L07. Indexed event arguments (fixed)

In the events defined in the **IStakingManagement.sol** file, consider marking `planId` argument as `indexed` in order to make offchain event tracking easier.

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

L08. Message sender variable (fixed)

In the **BatchTimelock.sol**, `onlyReceiver` modifier uses `_msgSender()` function for obtaining transaction sender address. However, in other parts of the contract `msg.sender` variable is used for the same purpose. We recommend obtaining message sender consistently across the codebase.

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

L09. Misleading variable name (fixed)

The project uses `MAX_APY` variable in two different ways. This variable contains the maximum APY value, which is described in its name. However, in `calculateStakeEarnings` function, `MAX_APY` represents the value of 100%. Currently, in the project, the maximum APY value equals 100%, and everything works correctly. However, we recommend separating the maximum APY value from 100% value.

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

L10. Missing check (commented)

The **BatchTimelock** contract contains `_addTimelock` function that contains checks of the parameters of the newly added timelock. We recommend adding a check for the `timelockFrom` variable in addition to existing ones.

> *Comment from the developers: The issue is fixed, we added check for non-zero value. There is not problem if time lock starts before current block timestamp , because we promised that time lock would start at the same time with IQT token launch, which is already launched*

The developers added the check `if (timelockFrom < 0) revert InvalidTimelockStart();`, which is redundant for variables of type `uint`. Therefore, we did not mark the issue as fixed.

L11. Rewards for terminated accounts (commented)

In the current version of the code, in the **BatchTimelock** contract an owner can still revert timelock termination by calling `determinate` function even after the `terminationFrom` time. As a result, the user can still receive tokens for the period when they were terminated. We recommend either clarifying this scenario in the documentation or changing the code behavior.

> *Comment from the developers: There is no issue here. The functionality is following,* `terminate` *and* `terminationFrom` *are defining the period from which the person won't be capable to withdraw more tokens than vested amount till* `terminationFrom` *period, but will have a possibility to withdraw them. The use case is that Timelock will be used for employee pool, if person quits company we have a notice period of 2 months, so* `terminationFrom` *would be* `now + 2 months` *and after quitting working the company person will have a possibility to withdraw, but only till the moment he/she was working with us.*

L12. Setter does not emit an event (fixed)

In the **StakingManagement** contract, `setStakingLimits`, `setMininumStake` and `setMaximumStake` setters do not emit events. We recommend adding them in order to make off-chain monitoring easier.

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

L13. Storage packing (fixed)

Consider placing `staker` and `withdrawn` fields near to each other inside `Stake` struct in order to reduce storage slots consumption.

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

L14. Storage packing (fixed)

In `Timelock` struct, consider packing the `receiver` field with the `isTerminated` one to reduce storage consumption.

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

L15. Unchecked return value of set add/remove functions (fixed)

The methods `EnumerableSet.add` and `EnumerableSet.remove` return a boolean flag indicating the success of the operation. It is important to validate the return value of these methods, as it helps to prevent unexpected scenarios such as adding an existing element or removing an element that is not present in the set.

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

L16. Unchecked transfer results (fixed)

The results of `_stakingToken` transfers are not currently validated. While the **IQT** token reverts transactions with incorrect transfers, it is considered good practice to validate the return values of `transfer`/`transferFrom` functions for `ERC20` tokens.

The same issue is present in the **BatchTimelock** contract.

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

L17. Redundant code (fixed)

In the **BatchTimelock** contract at line 188 function there is a redundant check. According to the previous code, `vestedPortion` is always greater or equal to `lock.releasedAmount`.

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

L18. Redundant code (fixed)

In the **BatchTimelock** contract, the code at lines 192-194 is redundant. It does not matter whether the `if` condition is `true` or `false`; the `claimable` variable is returned in both cases.

The issue has been fixed in commit 6b73be6d8f37d2d63eac117ce1b095cceebda84a.

L19. Redundant code (fixed)

`_checkAllowance` function in the **Staking** contract is redundant. The allowance of the user to the contract is automatically checked during the token transfers.

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

L20. Redundant code (fixed)

In the **Staking** contract in the `calculateStakeEarnings` function, the code at lines 167-169 is redundant. In case when the if case is true, the `compoundingPeriods` variable equals to `stakeRecord.endTimestamp - stakeRecord.startTimestamp` according to lines 155-157.

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

Notes

N01. Some storage variables are not required for the correct operation of the system (fixed)

In the project, there are several storage variables that can be calculated off-chain. Hence, storing such variables for the correct operation of the smart contracts is not required. Usually, we recommend removing such variables in order to reduce gas costs. However, the developers want to retain them so that these variables will be easily accessible in the future in case of unpredictable events, such as developers changes, new codebase, internal refactors, etc. This approach results in higher transaction costs but requires less restrictive project maintenance.

> *Comment from the developers: The issue is not relevant as we decided to change the deployment destination chain from Etheereum Mainnet to Polygon Mainnet. So most of gas related problems are not so significant.*

N02. Possible frontrunning

The **StakingManagement** contract allows the removal of a staking plan by calling the `removeStakingPlan` function. As a fix for the M06 issue, the developers added a check that the plan can only be deleted when it does not have any users. However, a malicious user can deposit funds right before the `removeStakingPlan` function is called, causing its execution to revert.

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Oleg Bobrov, Junior Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager

December 6, 2023