IQ Protocol Security Analysis

by Pessimistic

This report is public

July 15, 2022

Abstract

In this report, we consider the security of smart contracts of IQ Protocol 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 Protocol smart contracts. We performed our audit according to the procedure described below.

The audit showed only several issues of low severity. They do not endanger project security.

After the initial audit, the codebase was updated. In this update, all low severity issues were fixed. Also, the developers provided informative comments on the notes.

General recommendations

We have no further recommendations.

Project overview

Project description

For the audit, we were provided with IQ Protocol project on a public GitHub repository, commit cb95b474552d86b21071560e597385793fefadf8.

The scope of the audit includes the whole repository.

The documentation for the project included private documents and the following links:

- IQ Rental Marketplace
- Warper

All 345 tests pass successfully. The code coverage is 93.97%.

The total LOC of audited sources is 2832.

Codebase update #1

For the recheck #1, we were provided with IQ Protocol project on a public GitHub repository, commit a65d466b7f7728c40bd8840024bc41b4ca52e3ca.

In this update, all issues were fixed. Also, new tests were added to the project. All 355 tests passed successfully. However, the code coverage decreased to 93.5%.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan the project's codebase with the automated tool Slither.
  - We manually verify (reject or confirm) all the issues found by the tool.
- Manual audit
  - We manually analyze the codebase for security vulnerabilities.
  - We assess the overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

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.

The audit showed no critical issues.

Medium severity issues

Medium 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.

The audit showed no issues of medium severity.

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. CEI pattern violation (fixed)

Some functions within the project violate CEI pattern. We highly recommend following CEI pattern to increase the predictability of the code execution and protect from some types of re-entrancy attacks.

1. The `listAsset` function of **Metahub** contract performs an external call at line 226 and then writes the storage variables at line 239.
2. The `rent` function of **Metahub** contract performs an external call at lines 330. After this call, the function changes the storage at lines 351–357.
3. The `handleRentalPayment` function of **Accounts** contract performs an external call at line 111 and then writes to storage at line 118-127.

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

L02. Naming collisions (fixed)

Using the same names for different constants, variables, or functions degrades code readability. In `warp` function of **ERC721WarperController** contract, the returned value's name `collectionId` intersects with the function's name of the parent contract **ERC721AssetController** at line 65.

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

L03. Function visibility (fixed)

Consider declaring the following functions as `external` instead of `public` to improve code readability and optimize gas consumption:

1. The `metahub` function of **AssetVault** contract.
2. The `isRecovery` function of **AssetVault** contract.
3. The `mint` function of **ERC721Warper** contract.
4. The `rentalBalance` function of **ERC721WarperController** contract.

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

L04. Literals (fixed)

Consider declaring value `10_000` at lines 338 and 341 of **Rentings** contract as a `constant` to improve code readability and maintainability.

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

Notes

N01. Restricted ERC721 implementation

The **ERC721Warper** contract inherits from **IERC721** interface. However, the contract restricts certain functionality of the token: the `approve` method is not allowed, all the transfers are controlled by **Metahub** contract, etc.

> *Comment from the developers: A life-cycle of a warped asset is bound to the underlying rental agreement. The renter is considered to be the owner of a particular warped asset during the entire period of renting. The Metahub contract is responsible for managing the rental agreements and ensure their automatic expiration, thereby revoking the warped asset ownership from a renter after the rental agreement expiration. A renter cannot approve or set operator for the warped asset he owns, since the Metahub is the only entity that can move warped assets. Some restrictions might be lifted in future, if the protocol implements the concept of transferring rental agreements.*

N02. Call to an external contract

The `warp` function of `ERC721WarperController` contract calls `IERC721Warper(warper).mint` at line 39 or `_transferAsset` at line 41. Both these methods make `IERC721Receiver(to).onERC721Received()` call to an external contract. The receiving contract can have any logic.

> *Comment from the developers: The goal is to make Warper contract behavior closer to industry standards, therefore we adhere the standard recommendations regarding the onERC721Received function call. Indeed the incorrect behavior of the warped asset recipient might break the renting flow and cause a transaction revert. Although the affect of incorrect and/or malicious behavior of the warped asset recipient should be limited to a single renting agreement operation and should not affect other participants.*

N03. Overpowered roles

The system has the following roles that can affect the project's operation:

1. The `ADMIN` role can:
    - Register listing strategies.
    - Switch the **AssetVault** contract to recovery mode that allows delisting assets.
    - Withdraw `ERC20` tokens from the vault. This could be a problem for `ERC721` tokens that have a non-standard transfer since the token address is passed as a parameter. This is only possible if the key of this role is compromised.
    - Change the controller of any warper.

2. The `SUPERVISOR` role can:
    - Pause/unpause asset vault.
    - Change listing controller.
    - Add, remove, enable, disable presets.

There are scenarios that can lead to undesirable consequences for the project and its users, e.g., if private keys for any of these roles become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from the developers: The ADMIN role will be delegated to a multisig account.*

N04. Limitations on NFT usage

The system does not properly handle NFTs that suppose to accrue rewards to the owner. Since the tokens are stored on the **AssetVault** contract, it will accrue all the rewards. However, the contract does not have any functionality for transferring these rewards to users. Thus, all the rewards will be stuck on the **AssetVault** contract.

> *Comment from the developers: Should any reward accrue during the original asset listing period, it will be accumulated on the corresponding AssetVault contract as the contract will be effectively an owner of the asset. The IAssetVault interface exposes the withdrawERC20Tokens which allows the protocol admin to withdraw ERC20 accumulated tokens. This mechanism can be used by admin to withdraw reward and re-distribute it to the original asset owners.*

N05. Overcomplicated code

The codebase implements functionality for using both ERC721 and ERC1155 tokens. As a trade-off for this flexibility, the code becomes more complicated. However, ERC1155 tokens are not used within the project.

> *Comment from the developers: Many trade-offs were made to ensure enough flexibility for future protocol extension. It allows to add ERC1155 token support, as well as other non-standard implementations i.e. CryptoPunks in future. Listing strategies and Warper mechanics allow to extend the protocol functionality to accommodate new use-cases.*

This analysis was performed by Pessimistic:

Daria Korepanova, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Ivan Gladkikh, Junior Security Engineer
Nikita Kirillov, Junior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

July 15, 2022