IQ Protocol V2 Security Analysis

by Pessimistic

This report is public

December 30, 2022

Abstract

In this report, we consider the security of smart contracts of IQ Protocol V2 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 V2 smart contracts. We described the audit process in the section below.

The audit showed two critical issues: Accessing a non-allocated array and Accessing a non-existing array element. We also found several issues of medium severity, including Passing a wrong argument, Using a wrong index, Incorrect conditions, Invalidated mechanic support, etc. Moreover, many low severity issues were found.

After the initial audit, the codebase was updated. This update included the new functionality. And we found the new case of Accessing a non-existing array element issue of critical severity, Confusion with the use of contract addresses issue of medium severity and other low severity issues. Also, we updated Insufficient code coverage with tests issue of medium severity. The number of tests increased.

After the first recheck, the codebase was updated. In this update, one critical Accessing a non-existing array element, two medium (Invalidated mechanic support, Confusion with the use of contract addresses), and most of the low severity issues were fixed. This update included a new functionality. And we found one new low severity issue. The number of tests and code coverage increased. Also, Insufficient code coverage with tests issue of medium severity was updated.

After the second recheck, the codebase was updated again. In this update, the developers fixed Insufficient code coverage with tests issue since they increased the number of tests. Also, the developers fixed one low severity issue and commented on several others.

The project has incomplete documentation.

General recommendations

We recommend supplementing the documentation.

Project overview

Project description

For the audit, we were provided with IQ Protocol V2 project on a public GitHub repository. We previously audited this repository on commit a65d466b7f7728c40bd8840024bc41b4ca52e3ca.

The scope of this audit included:

- Changes to the `main` branch on commit f9a7c30a34aaa58ab2949c004e796a0e3c3414b1;
- **contracts/listing/** and **contracts/tax/** folders on commit d51c189c6b241d1f4cabc04249c9755f03702ce7 (`listing-configurator-with-bundles` branch);

During the audit, the developers performed several updates with new functionality and fixes for reported issues. Thus, we added the commit 140230a2605c0a10b2199e3cfef924ac7d0b0e30 to the scope of the audit. On this commit, we analyzed:

- Fixes in **contracts/listing/** and **contracts/tax/** folders;
- **contracts/listing/listing-configurator/** folder;
- **contracts/contract-registry/ContractEntity.sol** file;
- **contracts/metahub/Metahub.sol** file;
- dependencies.

The documentation for the project included https://docs.iq.space/nft/general/iq-space link. However, it does not cover Listing Configurators logic.

All 156 tests pass successfully. The overall code coverage is 47.7%. Tests do not cover important scenarios, e.g., registration of listing logic, tax terms, withdrawal of listing assets, etc.

Codebase update #1

After the initial audit, the codebase was updated. For the next part of the audit, we were provided with commit 36de28b3ee0a8afae5604f87c69976cded7660e5 (`wizards-with-listing-configurator-presets branch` branch).

The audit scope included the logic of wizards and the logic involving payment tokens.

194 tests out of 320 passed, and 126 tests had pending status. The code coverage was 42.67%.

This update did not include fixes. During the audit, we discovered one issue of critical severity, one issue of medium severity, and a few issues of low severity.

Codebase update #2

After the recheck #1, the codebase was updated, and we were provided with commit 58066920035d3a18c93f65d3398a21cdacb2605e (`pre-release-tests-by-alan` branch).

The audit scope included the logic of the reward distributor.

395 tests out of 463 passed, and 68 tests had pending status. The code coverage was 58.4%.

In the update, the developers fixed one critical issue and two medium issues. Also, new tests were added.
However, during the audit we discovered one issue of low severity.

Codebase update #3

After the recheck #2, the codebase was updated, and we were provided with commit 58e4464e3c38bfd8fc5d2620a1c8590dafcf4671 (`pre-release-tests-by-alan-v2` branch).

The scope included all contracts.

412 tests out of 480 passed, and 68 tests had pending status. The code coverage was 81.73%.

In the update, the developers fixed one medium and low severity issues. Also, new tests were added.

Audit process

We started the audit on October 17 and finished on November 8, 2022.

We previously audited the project's repository on commit `a65d466b7f7728c40bd8840024bc41b4ca52e3ca`. So, this audit included three stages:

- First, we analyzed the changes on the `main` branch.
- Then, we inspected the new functionality on `listing-configurator-with-bundles` branch.
- Finally, we analyzed the `wizards-with-listing-configurator-presets` branch. This branch included new functionality and fixes for some of the issues reported during the previous stages.

We discussed with the developers the current state of the project and highlight those parts of the code and logic that require additional attention during an audit:

- Whether users can make `delegatecall`s to withdraw money from the vault or rewards through the **Metahub** contract, perform a self-destruct, or any other malicious actions.
- Users-added contracts: are they isolated? Can they withdraw users' tokens or lock the system?
- Recalculation of commissions paid by the renter.
- New functionality with listing and tax terms and listing configurators.

Within the audit, we stayed in touch with the developers and discussed confusing or suspicious parts of the code, and reported all the occurrences after each stage.

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:

- Can the configurator steal users' tokens, withdraw money from the system or block the protocol?
- Whether the example of presets is properly integrated
- Are the roles issued correctly? Is the case when no one can get the role, or can anyone get it possible?
- Can users substitute the controller for the configurator to their own?
- Is it possible to front-run a rental?
- Is it possible to withdraw money from the vault?
- Are the listing and tax terms correctly defined according to the description of their levels?
- Is it possible to pass different assets to the function of withdrawing listed assets and steal them from the vault through the Metahub contract.
- Is the initialization and use of listing and tax terms for calculating rewards for the lister, the protocol, and the universe matching the description?

We scanned the project with the static analyzer Slither with our own set of rules and then manually verified all the occurrences found by the tool.

We ran tests and calculated the code coverage.

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

After the first part of the audit, we received the update on `wizards-with-listing-configurator-presets` branch. We started to make the audit on November 14 and finished on November 22, 2022.

Before the beginning, we had a call with the developers, who explained the features and difficult moments of the new functionality.

During the audit, we kept in touch with the team of IQ Protocol and discussed unclear parts of the code.

In addition, we made the following checks:

- There are no mistakes in the process of transferring logic from **Metahub** to **RentingManager;**
- The correct roles protect the functions;
- Base tokens are correctly converted to payment tokens;
- Replay attacks are impossible;
- The setup process in the wizards is correct;

At the end, we scanned the project again with the static analyzer Slither with our own set of rules and then manually verified all the occurrences found by the tool.

We also ran tests, calculated the code coverage of the new scope, and updated the report.

For the next audit, we received the update on `pre-release-tests-by-alan` branch. We started to make the audit on December 6 and finished on December 13, 2022.

Firstly, we checked the fixes of the issues from the previous audit. And after that, we started to check the new functionality.

During the audit, we asked the developers questions and resolved incomprehensible points in the code.

Also, we made the following checks:

- Is it possible to withdraw tokens from **ERC20RewardDistributor** contract?
- Is the percentage of rewards sent to `universe`, `protocol`, `lister`, `renter` counted correctly?
- What happens if one recipient specifies a 100% fee?

At the end, we scanned the project again with the static analyzer Slither with our own set of rules and then manually verified all the occurrences found by the tool.

We also ran tests, calculated the code coverage of the new scope, and updated the report.

For the next audit, we received the update on `pre-release-tests-by-alan-v2` branch. We started to make the audit on December 28 and finished on December 29, 2022.

Firstly, we checked the fixes and code refactoring. After that, we ran tests and calculated the code coverage.

At the end, we updated the report and added comments from the developers.

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. Accessing a non-allocated array (fixed)

In **ListingManager** contract, the `createListing` function tries to access elements of `listing.assets` at line 126 before its allocation and, therefore, always reverts. Consider allocating the array using the `new` keyword.

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

C02. Accessing a non-existing array element (fixed)

The following functions attempt to assign an array element before updating the array's length:

- The `register` function of **Listings** contract at line 221.
- The `register` function of **Rentings** contract at line 297.

> *This part of the issue has appeared on the commit `36de28b3ee0a8afae5604f87c69976cded7660e5`.*

According to Solidity documentation, accessing an array past its end causes a failing assertion. Consider updating the array's length using the `push` method before assigning its elements.

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

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.

M01. Passing a wrong argument (fixed)

In **ListingTermsRegistry** contract, the `allListingTerms` function should pass `_globalListingTerms` mapping instead of `_universeListingTerms` as an argument when calling `_paginateIndexedListingTerms` function at line 206.

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

M02. Using a wrong index (fixed)

In **ListingTermsRegistry** contract, the `_paginateIndexedListingTerms` function attempts to access `i`th element of the `_listingTerms` mapping at line 317. However, the function should access an element at the `listingTermsIds[i]` index instead.

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

M03. Incorrect conditions (fixed)

In **ListingTermsRegistry** contract, the `areRegisteredListingTermsWithParams` function can return a wrong result due to incorrect conditions at lines 344, 346, and 379.

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

M04. Invalidated mechanic support (fixed)

The **GeneralGuildListingConfiguratorPreset** contract implements the **ICanListAssets** interface. However, the contract does not override the `supportsInterface` function of the **AbstractListingConfigurator** `abstract` contract. As a result, the `_validateListingMechanics` function of the **ListingConfiguratorController** contract treats the **GeneralGuildListingConfiguratorPreset** contract as if it does not support the **ICanListAssets** mechanic and, therefore, does not call the `__canListAssets` function of the configurator preset.

Consider implementing the `supportsInterface` function within the **GeneralGuildListingConfiguratorPreset** contract.

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

M05. Insufficient code coverage with tests (fixed)

The project has tests; all 156 tests pass successfully. However, the code coverage is below 50%, and tests do not cover important cases of contracts' behavior, e.g., registration of listing logic, tax terms, withdrawal of listing assets, etc.

Testing is crucial for code security, and an audit does not replace tests in any way. We highly recommend covering the code base with tests and making sure that all tests pass and that the code coverage is sufficient.

> *The update on the commit `36de28b3ee0a8afae5604f87c69976cded7660e5`:*
> *The number of tests has increased to 194. The code coverage of the current scope is below 50%. The tests have not covered the new functionality of payment tokens and the part of the wizards. They also need to cover the logic of updated renting.*
>
> *The update on the commit `58066920035d3a18c93f65d3398a21cdacb2605e`:*
> *The number of tests has increased to 395. Even though the code coverage of the current scope has increased to 58.4%, some parts of the codebase are barely covered with tests, such as `renting`, `wizards`, etc.*

The issue has been fixed and is not present in the latest version of the code.The entire code coverage is 81.73%.

M06. Incomplete documentation

The project has documentation. However, the documentation is incomplete and does not describe the listing configurator logic and other important parts of the project.

Proper documentation should explicitly explain the purpose and behavior of the contracts, their interactions, and key design choices. It is also essential for any further integrations.

M07. Confusion with the use of contract addresses (fixed)

All over the code, the addresses of the **Metahub** and **RentingManager** contracts get mixed up. The examples are below:

- `rent`, `_handleRentalPayment`, `estimateRent` functions of the **RentingManager** contract always revert since they use `address(this)` as the Metahub address. However, in this case, it is the address of the **RentingManager** contract.
- In the **ERC721Warper** contract, there are cases when the **Metahub** address should be replaced with the address of the **RentingManager** contract to avoid the revert:
    - `onlyMetahub` modifier in `_beforeTokenTransfer` function;
    - the returned value of the `_metahub()` function which is called in `_getWarperRentalStatus`, `balanceOf`, `ownerOf`, `getApproved`, `isApprovedForAll` functions.

    > *It does not matter who will be the default owner of a warped asset in the `ownerOf` function, since `ERC721Warper._transfer` is overridden (it overwrites the new owner), and the `ERC721Warper._beforeTokenTransfer` function has the restriction that only **RentingManager** can transfer warped assets.*

- **RentingManager** address should be passed as an argument instead of **Metahub** at line 98 in the `validateRentingParams` function of the **ERC721WarperController** contract.

The issues have been fixed and are 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 versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Redundant code (fixed)

The project contains redundant code that affects readability and increases gas consumption:

1. The **ListingManager** contract declares `_strategyRegistry` variable at line 78 but does not use it.
2. In **Metahub** contract, the `registerAsset` function verifies that the asset is not registered at line 103. If not, it attempts to register the asset with the `Assets.registerAssets` call. However, the `registerAssets` function of **Assets** also includes the check of whether the asset is registered at line 119. Consider removing the redundant check at line 103 of **Metahub** contract.
3. The **ListingConfiguratorPresetFactoryStorage** and **ListingConfiguratorPresetFactory** contracts import the same library, **EnumerableSetUpgradeable**. However, the **ListingConfiguratorPresetFactoryStorage** contract is inherited by the **ListingConfiguratorPresetFactory** contract.

   Consider removing this import from the **ListingConfiguratorPresetFactory** contract.
4. The **ListingConfiguratorRegistry** contract declares the `onlyListingConfigratorAdmin` modifier at line 66. However, this modifier is not used within the project.
5. In **FixedRateWithRewardListingController** and **FixedRateListingController** contracts, the `listingTerms.strategyId != strategyId()` check in `calculateRentalFee` function is redundant since it duplicates the check inside the `compatibleStrategy` modifier of the `decodeStrategyParams` function.
6. `listingId` variable is never used in `onlyAuthorizedToAlterListingTerms` modifier of the **ListingTermsRegistry** contract.
7. `_rentingRegistry` variable of **MetahubStorage** is not used.
8. `duration` function of the **Rentings** is not used.
9. The `onlyTokenQuoteManager` modifier in **TokenQuote** contract is never used.
10. In **ERC20RewardDistributor** contract, there are unused imports at lines 5, 8, and 17.

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

L02. Redundant using...for directives (fixed)

The codebase contains multiple redundant `using...for` directives. Consider removing them to improve code readability and optimize gas consumption:

1. In **ListingTermsRegistryStorage** contract at lines 15,16: the contract **ListingTermsRegistry** already contains these directives.
2. In **ListingTermsRegistry** contract at lines 23 and 26.
3. In **TaxTermsRegistry** contract at line 25.
4. In **ContractEntity** contract at line 12.

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

L03. Reference type optimization (fixed)

Consider declaring reference type arguments of `external` functions as `calldata` instead of `memory` to save gas. There are multiple occurrences of this issue all over the code in the project.

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

L04. Extra reading from the storage (fixed)

The **Listings** contract reads `listingRecord.assets[i].token()` values inside `for` loops. However, this value remains the same within the loop since all the assets belong to the same collection. Consider reading the value only once outside the loop:

1. In the `register` function at line 222;
2. In the `remove` function at line 240.

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

L05. Misleading error name (fixed)

In **Metahub** contract, the `onlyListingManager` and `onlyRentingManager` modifiers can revert with the `CallerIsNotWarperManager` error. However, in both cases, the error name is misleading.

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

L06. Misleading event name (fixed)

In **ListingConfiguratorPresetFactory** contract, consider renaming the `ListingConfiguratorPresetEnabled` event to `ListingConfiguratorPresetDisabled` in `disablePreset` function.

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

L07. Inefficient mappings usage

Accessing a mapping element is an expensive operation since it requires `keccak256` hash calculation for the element's key. Thus, when the code needs access to the same mapping value multiple times, consider storing this element to a local variable to optimize gas consumption:

1. **(fixed)** In **Listings** contract, the `register` function reads `self.listings[listingId]` value at lines 211-217 and 221.
2. In **Listings** contract at line 244, the `remove` function accesses the `self.listings[listingId]` value instead of `listingRecord` to delete an element.
3. **(fixed)** In **Assets** contract, the `returnAssetFromVault` function reads `self.assets[assetToken]` at lines 201-202.
4. **(fixed)** In **Assets** contract, the `transferAssetToVault` function reads the `self.assets[assetToken]` value at lines 189-190.
5. **(fixed)** In **Rentings** contract, the `register` and `updateAgreementConfig` functions read the `self.agreements[rentalId]` value at lines 280-287, 297 and 320-324.
6. **(fixed)** In **Rentings** contract, the `register` function reads the `self.renters[agreement.renter]` value at lines 305, 307.

Notes

N01. Overpowered roles (commented)

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

1. The `ADMIN` role can:
    - Register listing and tax strategies.
    - Change the controller of any configurator and warper.
    - Register contracts in **Metahub** contract.
    - Register and remove tax terms.
    - Grant `LISTING_WIZARD`, `UNIVERSE_WIZARD`, `TOKEN_QUOTE_SIGNER` roles.

2. The `SUPERVISOR` role can:
    - Change listing and tax controllers.
    - Add, remove, enable, and disable presets of the listing configurator and warper.

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.*

N02. Draining reward distributor out of funds (commented)

If a warper transfers rewards to **ERC20RewardDistributor** contract and does not immediately call `ERC20RewardDistributor.distributeExternalReward` function in the same transaction, the following exploit can occur:

- Assume there are `N` tokens `T` on **ERC20RewardDistributor** contract address.
- Attacker selects an arbitrary listing open available for renting. Assume it has `id=i`. This listing should also have registered listing terms with fixed rate strategy. Assume it has `id=t_i`.
- Then he calls `RentingManager.rent()` with `paymentToken` equal to base token address, `renter` equal to attacker's controlled address, `rentalPeriod=1`, `listingTermsId=t_i`, warper equal to corresponding warper. Saves the returned `rentalId` as `r_i`.
- As a result, `N` tokens `T` will be transferred from **ERC20RewardDistributor** to the `renter` address (line 227 in **Accounts.sol**).
- The only restriction on this exploit is the amount of tokens on the **ERC20RewardDistributor** contract, meaning that `distributeExternalReward` can be called multiple times as long as there are enough tokens.

> *Comment from the developers: This is a known potential issue, but:*
>
> - *the IQ Protocol is not planned to have any funds allocated on it;*
> - *the usage of this contract comes with a strict guide, where transferring funds without calling certain function is an erroneous behavior.*

This analysis was performed by Pessimistic:

Daria Korepanova, Security Engineer
Ivan Gladkikh, Junior Security Engineer
Yhtyyar Sahatov, Junior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

December 30, 2022