Intentable's Smart Transaction technology powered by the Kirobo FCT Platform

Smart Wallet Security Analysis

by Pessimistic

This report is public

August 15, 2022

Abstract

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

This report combines the results of two audits for different scopes of Kirobo repository.

The audit showed one critical Access control vulnerability and several medium severity issues, including Token price manipulation, Non-transferable NFTs, Bad condition, Predictable NFT value, Reverting token inheritance, etc. Also, many issues of low severity were found. Some tests did not pass; the code coverage was not measured.

After the initial audit, the developers performed several codebase updates. In these updates, they fixed most of the issues, added new functionality, and implemented new tests.

General recommendations

We recommend implementing CI to run tests and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Intentable project on a public GitHub repository, commit b7cf11150d101c6b6b2e038aaee21703dc4ba089.

The scope of the audit included only:

- **contracts/RecoveryOracle.sol** file,
- **contracts/RecoveryWallet.sol** file,
- **contracts/RecoveryWalletCore.sol** file,
- **contracts/nft/** folder,
- dependencies.

The documentation for the project included:

- **Wallet, Oracle, Inheritance, Backup, Activator.pdf** file, sha1sum is `44618f46e1de44c1c67cfd8e1dbfad1508250988`,
- **Kirobo Collectibles White Paper.pdf** file, sha1sum is `f7916dec47277bd2255c64943fdd00eda95652e8`.

The codebase is partially covered with NatSpec comments. The developers also provided an additional explanation of the project during the call.

The project has 142 tests. However, four of them fail to pass. The code coverage is not measured.

The total LOC of audited sources is 2521.

Codebase update #1

After the initial audit, the codebase was updated. For the recheck, we were provided with commit db29315b623cfa0fb71af02723b8650505313704.

This update included fixes for most of the issues in the **contracts/nft/** folder mentioned in the initial audit, fixes for issues discovered by the developers, new functionality, and tests updates. The developers implemented the code coverage calculation.

The recheck did not show new issues in this update.

Codebase update #2

After recheck #1, the developers performed codebase update #2. For recheck #2, we were provided with commit b12645afce2a36e1480ec0e017891b77e706984b. This update included changes in the system design (mostly for the **Activator** logic), new tests, new functionality, and fixes for the issues mentioned in the report and discovered by the developers.

The number of tests increased up to 245. All tests pass.

Recheck #2 showed one new issue of low severity.

Codebase update #3

After recheck #2, the developers performed codebase update #3. For recheck #3, we were provided with commit 90ad169bbcd0cc795bd6466f0f3b7098f8bdab46. This update included additional fixes of the issues mentioned in the report.

Codebase update #4

After recheck #3, the developers performed codebase update #4. For recheck #4, we were provided with commit a1f723ced017ce14ee64ce09dad9bf87a5f6ec84. In this update, the developers fixed most of the low severity issues and added new tests. The number of tests increased up to 250. All tests pass successfully. The overall code coverage for the scope is 96.45%.

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.

C01. Access control vulnerability (fixed)

In the **RecoveryWallet** contract, functions `approveERC20`, `approveERC721`, and `approveERC1155` are public and have no modifiers for access restriction. Therefore, anyone can call these functions to approve any token amount to any address. A further call to `transferFrom` function from the provided address will transfer tokens from the contract to any wallet.

Consider adding the `onlyActiveOwner` modifier to these functions.

The issue has been fixed and is 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. Token price manipulation (fixed)

In **GasReturn** contract, `getKiroTokenPrice` function allows an attacker to manipulate the token price by sending KIRO/ETH to Uniswap pool. When the token price in ETH decreases, the amount of KIRO tokens for reward in the game increases. Using flash loans can amplify the effect of an attack. Users can also front-run those transactions that affect the token price. Consider using time-weighted average prices from Uniswap.

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

M02. Non-transferable NFTs (fixed)

In **KiroboNFT** contract, `_transfer` function does not implement transfer logic. As a result, Kirobo NFTs are non-transferable.

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

M03. Bad condition (fixed)

In **GasReturn** contract, the `closeMonths` function iterates through a `for`-loop while `i < (currentMonth - s_idToClosedMonth[i_nftId])`. However, `s_idToClosedMonth[i_nftId]` value changes on each iteration of the loop. As a result, an increase in the credit amount for the NFT is less than expected.

Consider storing the precalculated number of iterations to a local variable for further use in the loop condition.

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

M04. Tests issues (fixed)

The project has tests. However, four tests out of 142 fail to pass. The code coverage is not measured. The project includes issues that could be found in the testing process.

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

The developers added new tests. All tests pass in the latest version of the code.

M05. Predictable NFT value (fixed)

The values of not yet minted NFTs in **Minter** contract are predictable due to the following flaws in the system design:

1. The `random` function of **Minter** contract returns a value that can be calculated before the function call. Having an affordable result, users only need to make a transaction at a particular moment with proper gas cost, so it is mined in the block with the required number.

2. All the data required to find the gas return value is stored on the blockchain in private variables. If `s_key` and `s_nftAmountArray` variables appear on the blockchain before the end of the minting period, anyone can read them.

Therefore, there are two possible cases depending on when the activator writes `s_key` and `s_nftAmountArray` variables to the storage:

1. If the activator writes the variables before the end of the minting period, users can read them. Having all the necessary data and algorithms, users can precalculate values for NFTs and mint only the most valuable tokens.

2. If the activator writes the variables when the minting period has passed, some functionality will not work until the key is available.

> *Comment from the developers: Both sections mentioned (1 and 2) describe a potential issue revealing the NFT value before the mint, the planed flow is to mint and only after the mint process is over the reveal will take place. The reveal will happen only after the `s_key` parameter will be updated in the blockchain by Kirobo, so, there is no chance of figering out the value before hand. The **GasReturn** functionality only starts after the update of this `s_key` value.*

M06. Incompatible assets (fixed)

In the **Heritable** contract, `activateTokenInheritance` function performs arithmetical operations with incompatible assets at line 247:

```
totalTransfered = currentBalance - address(this).balance;
```

Here, `currentBalance` variable represents the amount of ERC20 tokens while `address(this).balance` is the contract's balance in Ether. Thus, `totalTransfered` represents an incorrect value.

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

M07. Reverting token inheritance (fixed)

In the **Heritable** contract, the `activateTokenInheritance` function attempts to perform a low-level transfer from `sp_heir.wallet` address at line 239 and thus reverts. The function should transfer from the `token` address. We recommend increasing test coverage to prevent such issues.

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

M08. ERC20 standard violation (fixed)

EIP-20 standard states:

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned!
```

However, the returned value of `approve` call is not checked (or returned) in `approveERC20` function of **RecoveryWallet** contract.

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 versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. CEI pattern violation

The following functions violate CEI pattern:

1. **(fixed)** The `mint` function of **Minter** contract. Consider calling `removeFromCollection` function before `sendValue` call and emitting an event at the end of the function.
2. **(fixed)** The `collectReward` function of **GasReturn** contract. Consider calling the `burn` function after writing to storage is complete.
3. The `activateTokenInheritance` function of the **Activator** contract. The call is safe since the KIRO address is trusted.

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

L02. Use of deprecated function (fixed)

The constructor of **KiroboNFT** contract uses `_setupRole` function. However, this function is considered deprecated. Consider using `_grantRole` function instead.

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

L03. Disregard a library documentation (fixed)

Consider declaring the `_beforeTokenTransfer` function of **KiroboNFT** contract as `virtual` to comply with Open Zeppelin documentation.

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

L04. Standard functionality reinvention (fixed)

Consider using well tested gas-efficient OpenZeppelin Base64 library instead of self-implementing standard functionality in `base64` function in **KiroboNFT** contract.

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

L05. Missing events (fixed)

Consider emitting events when important contract parameters change, especially if these parameters might be used off-chain. E.g., in `updateHoldingAmountNeeded` and `updatePricePerMonth` functions of **GasReturn** contract.

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

L06. Events arguments indexing (fixed)

Consider declaring `address` arguments of events as `indexed` to simplify further processing of these events:

1. In `TransferReceived` and `TransferSent` events of **GasReturn** contract at lines 77 and 78.
2. In `TransferSent` event of **Minter** contract at line 35.

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

L07. Dependency management (fixed)

1. In the **package.json** file, `openzeppelin-solidity` dependency points to the master branch. Consider declaring the exact version of the dependency to avoid sudden changes in the code when deploying.
2. The project contains **hardhat.config.js** file though it uses `truffle`.

> *Comment from the developers: The project was built using Truffle, and later on, we added Hardhat to get abilities regarding the solidity structures of the parameters of the contracts.*

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

L08. Redundant dependency (fixed)

The **KiroboNFT** contract instantiates **SafeERC20** trait at line 28. However, this trait is not used within the contract.

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

L09. Redundant import (fixed)

The **RecoveryWallet.sol** file imports **Heritable.sol** at line 10. However, this import is not used within the contract.

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

L10. Code duplication (fixed)

The `migrate` functions of **RecoveryWallet** and **RecoveryWalletCore** contracts are identical. Consider removing this function code from one of the contracts, as they proxy calls from one to another.

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

L11. Redundant type casting (fixed)

In **RecoveryWallet** contract, additional type casting from `address` is redundant at lines 126, 146, 159, 173, and 187.

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

L12. Redundant value

The `MAX_FEES` value is never used in the **Activator** contract.

L13. Redundant comparison

The boolean comparison at line 75 in the **Activator** contract is redundant.

L14. Excessive decomposition (fixed)

In the **Activator** contract, functions `_transferInheritancePayment` and `_transferTokenInheritancePayment` are private. The contract calls each of these functions only once. Thus, moving these blocks to separate functions complicates the code and weakens the overall function composition.

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

L15. Typos

The project contains many typos in names of functions and variables, in the comments, etc. E.g., in **GasReturn** contract, the `updatePricePerMonth` function updates the `s_prisePerMonth` mapping.

Some typos were fixed.

L16. Misspelled variable name (fixed)

The name of the `sentHairOK` variable in the **Heritable** contract has a typo. Consider renaming it to `sentHeirOK` at lines 239, 242, 276, and 280.

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

L17. Confusing naming (fixed)

GasReturn contract has 13 external functions with `onlyActivator` modifier. These functions set or update important parameters of the contract. The contract's constructor grants `ACTIVATOR_ROLE` to `msg.sender` and to `activator` address. If these addresses are considered to be **RecoveryOracle** or **Activator** contracts, they should implement functionality to call **GasReturn** contract, but they do not. If the `activator` address is an EOA, consider renaming **Activator** contract or `ACTIVATOR_ROLE` to avoid confusion.

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

L18. Missed payable keyword (fixed)

In the **RecoveryWallet** contract, consider declaring `to` argument of `execute2` function as `payable` since it accepts Ether.

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

L19. Gas consumption (fixed)

Consider declaring the following variables as `constant`s to optimize gas consumption:

1. `s_percentgeConst` variable at line 40 of **GasReturn** contract.
2. `s_baseURI` variable at line 29 of **Minter** contract.

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

L20. Gas consumption (fixed)

The **KiroboNFT** contract inherits from **ERC721URIStorage** OpenZeppelin library. However, the contract implements the logic for storing token URIs (including the `s_idToUri` mapping) that duplicates the logic from **ERC721URIStorage** library.

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

L21. Gas consumption (fixed)

In **GasReturn** contract, `revealGasReturn` function performs multiple storage operations on each iteration of the `for`-loop. This function consumes significant amounts of gas since storage operations are expensive. Consider calculating NFT hashes off-chain and passing an index of the required element to `revealGasReturn` function to decrease gas consumption.

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

L22. Variables visibility (fixed)

Consider declaring visibility for storage variables explicitly to improve code readability:

1. In **KiroboNFT** contract:
    - `s_gasReturnContract` variable at line 36.
    - `s_idToUri` mapping at line 40.
2. In **Minter** contract:
    - `s_idToUri` mapping at line 31.
    - `s_whitelist` mapping at line 32.
    - `s_kirobros` mapping at line 33.
3. In **GasReturn** contract, 11 mappings at lines 65–75.

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

L23. Functions visibility

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

1. **KiroboNFT** contract.
2. **RecoveryWallet** contract.
3. Functions `updateTokenInheritanceFunds`, `updateAllInheritanceFunds`, and `updateAllFunds` of **Activator** contract.

The developers updated visibility for some of the functions.

Notes

N01. Overpowered roles

The system has special roles with excessive powers:

1. The `MINTER_ROLE` of **Minter** contract can:
    - Change parameters for NFT minting: prices, deadlines, minting start moment (i.e., when minting becomes possible), etc.
    - Add NFTs to collection / remove NTFs from collection.
    - Add users to / remove users from the whitelist.
    - Add members to `s_kirobros` list / remove members from `s_kirobros` list.
    - Withdraw ETH from the contract to any address.

2. The `ACTIVATOR_ROLE` of **GasReturn** contract can:
    - Change the address of NFT contract.
    - Manipulate the mapping of NFT hashes, including adding empty values.
    - Change prise per month.
    - Change parameters of the contract that can affect token price and reward amounts.
    - Add users to whitelist / remove users from whitelist.

    `ACTIVATOR_ROLE` has been replaced with `OPERATOR_ROLE` in the code.

3. The `DEFAULT_ADMIN_ROLE` of **KiroboNFT** contract can:
    - Change the address of **GasReturn** contract.
    - Grant `MINTER_ROLE` to any address.
    - Mint NFTs.
    - Change the penalty free date.

> *Comment from the developers: All the powers that the roles hold are correct and are needed for the correct operation of the NFT minting.*

N02. Optimization algorithm

In the **Heritable** contract, the `clearInheritance` function optimizes the gas consumption using `address(0)` as a separator. However, neither the documentation nor comments in the code explain this solution. Also, tests do not cover this functionality.

N03. Token price calculation

The `getKiroTokenPrice` function of **GasReturn** contract does not consider the slippage when calculating the token price (KIRO/ETH). We recommend using Uniswap functionality instead.

> *Comment from the developers:* *We do not want to consider the slippage at all, we only use the price as a way of knowing the value, we are not doing a swap. So, that is ok for us.*

This analysis was performed by Pessimistic:

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

August 15, 2022