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

Kirobo FCT Security Analysis

by Pessimistic

This report is public

June 25, 2024

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.

The audit showed no critical issues. We found several issues of medium severity, including Inconsistent logic of transaction blockability, Inconsistent price update logic, Incorrect protocol fee calculation, and several low-severity issues.

After the initial audit, the codebase was updated. All the issues were fixed, and a new version of the documentation was added. Additionally, one medium severity issue Incorrect calculation of the shares, and several low severity issues were discovered and fixed by the developers. The project quality has improved.

General recommendations

We recommend performing cosmetic codebase refactoring to enhance modularity and improve code readability.

Project overview

Project description

For the audit, we were provided with Intentable project on a public GitHub repository, commit b4ad34628272b5db09e5c4cda13f4122cf07c1d5.

The scope of the audit included only the following files:

- **FCT_BatchMultiSig.sol**,
- **FCT_Controller.sol**,
- **FCT_Constants.sol**,
- **FCT_Helpers.sol**,
- **PureValidator.sol**,
- **PureSafeMath.sol**,
- **economy/FCT_Actuator.sol**,
- **economy/FCT_Tokenomics.sol**,
- **interfaces/IFCT_Actuator.sol**,
- **interfaces/IFCT_Controller.sol**,
- **interfaces/IFCT_Engine.sol**,
- **interfaces/IFCT_Tokenomics.sol**,
- **interfaces/IFCT_Runner.sol**,
- **RecoveryWallet.sol**.

The documentation for the project included:

- **LV-178192385-160922-0927.pdf** file, sha1sum is `21aded64cc126b3653a266a55fb130820d376cfe`,
- **LV-FCTBLOCKCHAINAPI-160922-0928.pdf** file, sha1sum is `2b852b2e14282207059930573fec572e3fad6b32`,
- **LV-FCTContracts-160922-0957.pdf** file, sha1sum is `1fa6673a66c7c534d38046733bf0f17d184122d4`,
- **LV-FCTTokenomics-160922-0928.pdf** file, sha1sum is `7b526576fcaf568d694dcd293c2c1db279026a2b`,
- **LV-WhatisFCT-160922-0931.pdf** file, sha1sum is `e60d708ae9874dad46f34ff650f185601d5175f9`.

We also set up a few calls with developers to discuss the details.

The total LOC of audited sources is 2829.

Codebase update

After the initial audit, the codebase was updated. We were provided with the commit f0f9e4ed3893577abc47ee8fd776229138126428. This update contains fixes for all issues. This version also fixes issues that were found by the developers independently (M05, L09, L10, L11).

Within this update, a new functionality was added to handle the usage of `ERC20` token under FCT more safely.

Besides, the project documentation has been updated. The new version of FCT description was presented.

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:

- Preliminary research
  - We read the documentation for the project.
  - We check whether the project has any integrations with third party contracts, and explore available APIs of these contracts.
  - We determine parts of the project that require extra attention during the audit.
- 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.
  - We verify that the behavior of the code corresponds the documentation.
- Report
  - We reflect all the gathered information in the report.

Inter alia, we verify that:

- The implementation of payout logic for activators and builders is correct.
- The parsing of the FCTs is accurate, and their execution is safe, especially transitions from one call to another.
- The price from UniswapV2 is calculated properly.
- No one can send transactions on behalf of the owner without the owner's signature.
- The activator cannot manipulate the gas price calculated by the contract.
- Gas usage is optimized, especially with storage operations.
- Encoding to EIP712 corresponds to the standard implementation.

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.

M01. Inconsistent logic of transaction blockability (fixed)

The implementation of the `fctIsBlocked` function in the **RecoveryWallet** contract evaluates transactions with `messageHash == 0` as non-blockable. However, the `batchMultiSigCall` function of the **FCT_BatchMultiSig** contract at line 566 assigns a zero value to the `messageHash` variable if the `FLAG_BLOCKABLE` bit is set to `1`.

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

M02. Inconsistent price update logic (fixed)

In the **FCT_Actuator** contract, the `s_lastUpdateDateOfPrice` variable is initialized with `block.timestamp` value at line 209. However, `activate` and `activateForFree` functions update it with `block.number`. Consider using consistent values for `s_lastUpdateDateOfPrice` variable and utilizing it only within `_updateKiroPrice` function for better code encapsulation.

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

M03. Incorrect protocol fee calculation (fixed)

In the **FCT_Actuator** contract, the `_chargeActivator` function assigns the wrong value to `s_balances[s_kirobo]` variable at line 653. Consider using the `activatorFees - builderPayment` remainder instead of `builderPayment` value.

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

M04. Insufficient design documentation (addressed)

The project has detailed documentation. Nevertheless, it lacks details regarding the design and implementation of the FCT scripting mini-language (e.g., variables and return values mechanics, conditional jumps, loops prohibition) and the activation process.

Proper documentation with an explicit explanation of the contract's mechanics, interactions, and design choices might reduce the probability of errors.

> *A new version of the project documentation is available.*

M05. Incorrect calculation of the shares (fixed)

The share was calculated from the overall price instead of the profit. So the activator got less money than it should (sometimes less than the actual transaction price)

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. Events with outdated values (fixed)

In the **FCT_Actuator** contract, `addFunds`, `addFundsTo`, `removeFunds`, `moveFundsToDeposit`, `deposit`, and `widthdraw` functions emit events with specific values and change these values afterward. As a result, these events contain outdated values.

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

L02. CEI pattern violation (fixed)

The **FCT_Actuator** contract violates the CEI pattern when making `safeTransferFrom` calls before updating storage variables at lines 302, 315, and 372. It does not threaten the security of the project since possible re-entry point is the Kiro token contract. However, we recommend following the CEI pattern in any case.

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

L03. Gas consumption (fixed)

1. In the `fctAllowedToExecute` function of the **RecoveryWallet** contract, consider moving `s_backup.state != BACKUP_STATE_ACTIVATED` check at line 344 outside the `for`-loop since this condition does not change its value during the iteration through `signers` array.
2. In the `calcPayments` function of the **FCT_Tokenomics** contract, `effectiveGasPrice` and `builderShare` fields of `payment` struct do not change during iteration through the `for`-loop at lines 182–197. Consider calculating them outside the loop to optimize gas consumption.

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

L04. Code clarity (fixed)

In the **FCT_Actuator** contract, the `_registerNonce` function checks and registers a nonce, but it also checks the activator's stake. This functionality contradicts the function's name. Consider moving the activator's stake check to a separate function to improve code readability and logic transparency.

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

L05. Implicit variables visibility (fixed)

In the **FCT_Actuator** contract, the visibility of the `s_lastUpdateDateOfPrice` and `s_timeBetweenKiroPriceUpdate` variables is not specified. We recommend declaring the visibility explicitly to improve code readability.

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

L06. Missing interface (fixed)

Consider declaring **RecoveryWallet** as the `IERC721Receiver` interface since it implements the `onERC721Received` function.

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

L07. Typos in NatSpec comments (fixed)

We mentioned the following typos in NatSpec comments:

1. In the **RecoveryWallet** at line 240: consider replacing `therfor can not` with `therefore cannot`.
2. In the **FCT_BatchMultiSig** contract at line 186: consider replacing `assosiatet` with `associated`.
3. Copypasted comment at line 187 in the **FCT_BatchMultiSig** contract does not apply to the corresponding return value.

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

L08. Typo in the function's name (fixed)

In the **FCT_Actuator** contract, the `widthdraw` function has a typo in its name. Consider renaming it to `withdraw`.

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

L09. Slippage in balances calculations (fixed)

There was a slippage in balances calculations, so over time, the total KIRO balance of users, builders and activators was different than the actual KIRO balance the **FCT_Actuator** holds.

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

L10. Wrong gas calculation for fees (fixed)

When using `activateForFree` or when there was no payer, the gas calculation for fees was wrong.

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

L11. Problems regarding the tokenomics in the Actuator (fixed)

There were two problems regarding the tokenomics in the Actuator:

1. It was operational before the first time period was over, giving unstable results.
2. It assumes that the first token in the pair is always Kiro which is not right on all networks. So on some networks, the calculation was wrong.

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

Notes

N01. Weak multisig (commented)

The optional external signers feature of the FCT implementation in the **FCT_BatchMultiSig** contract does not include signer uniqueness at lines 343–363. So the user can use multiple signatures from one signer to pass the multisig check.

This behavior is an intentional design choice to save gas.

> *Comment from the developers:* *wanted behavior. See documentation.*

N02. Hardcoded gas cost (fixed)

The **FCT_Actuator** contract uses pre-calculated costs for gas at lines 545–554. If execution costs of opcodes change with the new Ethereum hardforks, the contract will require re-deployment.

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

N03. EOA-only protection (fixed)

The `activate` and `activateForFree` functions of the **FCT_Actuator** contract use the `msg.sender == tx.origin` check to prohibit calls from smart contracts. This check protects from re-entrancy and flashloan-related attacks. However, some attacks might still be implemented via flashbots MEV-bundles.

From a design standpoint, contracts should work correctly independently of the calling account and safely interact with other contracts.

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

N04. Long method (commented)

Consider splitting the `batchMultiSigCall` function of the **FCT_BatchMultiSig** contract into several smaller functions to improve code structure and readability.

> *Comment from the developers:* *The method is long due to technical issues of "stack too deep" and gas optimizations. Even so, it was built with independent parts that could be considered as functions. Each part was documented with the `@notice` comment.*

This analysis was performed by Pessimistic:

Vladimir Tarasov, Security Engineer
Daria Korepanova, Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

June 25, 2024