INTENTABLE

Powered by KIROBO

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

FCT Economy Security Analysis

by Pessimistic

This report is public

October 2, 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 the FCT Economy submodule of Intentable smart contracts. We described the audit process in the section below.

The report has designations like C01, M05, L10, N08. The letter represents severity, and the number represents the issue number.

This is a condensed version of the report; a more detailed one can be found at the following links:

The original reports:

- Report #1. The fixed issues:
  - M02, M03, M05, L01, L02, L03, L04, L05, L08, L09, L10, L11, N02, N03, which were relevant to the current submodule;
  - M04 issue, which was addressed and was relevant to the whole project.
- Report #2. The fixed issues:
  - M01, M03, M06, M11, M12, L01, L02, L03, L05, L06, L10, N02, which were relevant to the current submodule;
  - L04, which was relevant for the whole project.
- Report #3. The fixed issues:
  - M04, M05, N06, which were relevant to the current submodule;
  - M01, which was addressed and was relevant to the whole project.

The report includes:

- Unresolved L01 (L17 in the Report #2);
- Notes with the `commented` status: N01 (N04 in the Report #2), N02 (N06 in the Report #2), N03 (N04 in the Report #3), and N04 (N05 in the Report #3).

All the tests passed. The code coverage is sufficient.

After the last recheck, which was fully described in Report #3 (see the Codebase update #8) and copied to the Project description, the codebase was updated again. The number of tests and the code coverage increased. We did not find any new issues.

During the small update, the developers fixed L01, which was copied from the Report #3, on commit 78c90bb9bf2b38c50b03ef3d89f0d125e2bc3d6a.

According to our recommendations, the developers split the long function in the **FCT_BatchMultiSig** contract, improving the code readability. However, it is still important to note that the project and its architecture are complex, though we realize that it is difficult or impossible to implement simply.

It is crucial to study the three reports above to get a full picture of the security of smart contracts.

General recommendations

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

Project overview

Project description

For the audit, we were provided with Intentable project on a private GitHub repository, commit cf31b27dd9e44e016ced8e3060a487e7e1e2d5cc.

It is a condensed version of the report. More detailed information about codebase updates can be found at the following links:

- Report #1;
- Report #2;
- Report #3.

The scope included:

- FCT_Actuator.sol;
- FCT_ActuatorCore.sol;
- FCT_ActuatorStorage.sol;
- FCT_Arch.sol;
- FCT_Arch_Mainnet.sol;
- FCT_Arch_Arbitrum.sol;
- FCT_Arch_Optimism.sol;
- FCT_Tokenomics.sol;
- FCT_Funding.sol;
- IFCT_Tokenomics.sol;
- IFCT_Actuator.sol;
- IFCT_Funding.sol.

The documentation for the project included https://kirobo.gitbook.io/fct-developers-guide/.

The total LOC of the audited scope is 1532.

All 592 tests passed. The code coverage of the project was 84.64%.

Codebase update #1

For the recheck, we were provided with Intentable project on a private GitHub repository, commit 9ba7299653676b06cadfcbfac2d1f9fff66d333c.

The developers provided the test results and the code coverage. All 630 tests passed, the code coverage of the scope was 94.53%, and the code coverage of the FCT platforms was 92.93%.

Audit process

We started to check the FCT platform around two years ago and created four extensive reports. The developers asked us to split these reports into seven submodules. Each submodule has:

- The last common commit for all reports;
- The same results of the tests and the code coverage for the whole project;
- The individual scope;
- Links to the corresponding previous reports;
- The list of fixed issues, e.g., M01, M04, L03 (see the description in the Summary), etc.;
- The descriptions of unfixed or commented issues that are still actual;
- The findings relevant to the whole project.

Issue descriptions copied from previous reports can have different contract names and lines, as they were checked again and updated due to the last commit in the Project description.

We started auditing the FCT platforms in 2022. During this time, we made three FCT reports and one report for the Smart Wallet part.

Each report is a continuation of the previous one. We made the split in the process to avoid one infinitely extensive report. The chronology of the reports:

- Smart Wallet report - finished on August 15, 2022;
- Report #1 - finished on November 17, 2022;
- Report #2 - finished on July 26, 2023, and last updated on January 16, 2024;
- Report #3 - finished on June 26, 2024.

See the previous reports for a more detailed description of the issues and process. The following rechecks related to a specific module will only appear in the corresponding report.

We made recheck #1 for this scope on June 26-27, 2024. It was the first recheck in this report but not the first one in the entire audit process (see the previous reports). The developers provided the test results and the code coverage, as we could not run them. This update included code refactoring and fixes for adding the new functionality.

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 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 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. Incorrect comment (fixed)

In the `setNativeProfitBPS` function of the **FCT_Funding** contract the `require` message duplicates the message in the `setSwapBPS` function.

The issue has been fixed on commit 78c90bb9bf2b38c50b03ef3d89f0d125e2bc3d6a.

Notes

N01. Very long functions (commented)

Consider splitting the `activate` function (at lines 188-469) from the **FCT_Actuator** contract into several smaller functions to improve code structure and readability.

> *Comment from the developers from the previous audit:*
> *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.*

N02. Overpowered roles (commented)

The project has a lot of roles. They have the following options:

- **FCT_Actuator**, **FCT_ActuatorCore** contracts:
  - `PROTECTOR_ROLE` can set the value of ETH penalty;
  - `DAO_ROLE` can set the addresses of **FCT_Tokenomics**, **FCT_Funding** contracts, freeze and pause the ability to pay in ETH, and set other parameters and limits for FCT activation;
  - `MANAGER_ROLE` can set builder, the project wallet address and set the token price;
- `DAO_ROLE` of **FCT_Tokenomics** contract can:
  - enable/disable free mode for FCT payers;
  - change parameters that influence payments and gas cost refunds.

Thus, if the admin's private keys are compromised, there may be scenarios that could lead to undesirable consequences for the project and its users.

> *Comment from the developers: Now we are using the same key for all, but in production we plan to add multiSig for that.*

N03. Hardcoded gas cost (commented)

The **Arch** contract uses pre-calculated gas costs. If the execution costs of opcodes change with the new hardforks, the contract will require re-deployment.

> *The **Arch** contract has been split to three contracts: **Arch_Arbitrum**, **Arch_Mainnet**, and **Arch_Optimism**. This issue is also relevant now for these new contracts.*

> *Comment from the developers: As designed - The tokenomics contract was designed to be easily replaced without influencing other contracts logic and there is no need for users to do migration of assets when such replacement takes place (see N04 comment for more info about the need to pre-calculate some gas costs).*

N04. Estimation of gas consumption (commented)

We did not accurately check the gas calculations, the constant values for which were derived empirically. However, the `Arch.estimateExtraGas` function has the `forFree` parameter. If it is `true`, then the estimated gas is higher.

This function estimates the gas consumption of the `FCT_Actuator._chargeActivator` function. When `forFree` is `true`, it means that `FCT_Actuator._chargeActivator` is called with a 0 builder address. It spends more gas than if it was non-zero. However, when the builder is 0, there is only one writing to the storage.

Consider replacing line 29 with `uint256 addon = forFree ? 0: 6000` in the **Arch** contract.

> *Comment from the developers:*
>
> *Calculating the total gas used for a tx inside a contract is tricky. There is no opcode to get the gas limit being used, only the gas left. In order to calculate a fair amount of gas that will be used to charge the user, there is a need to estimate the function call gas amount and the rest of the code being run after charging the user (by updating user's balance on the contract storage).*
>
> *Accurate calculation of function call is possible but not reasonable, because it depends on the number of '0' in the calldata. checking this costs more than the tx itself. Accurate calculation of the EVM code that runs after charging the user, is not possible.*
>
> *Our solution is to estimate both, adding this to the parts of the function execution we can calculate on-chain (checking the diff between two gas-left calls) We also emit these calculated values via events.*
>
> *In order to make sure our estimations are close enough (the user is paying at least 20% fees initially and much more in the future), we added test printouts that show the difference between calculated gas (found on tx events) and real gas consumption (from receipt). For most tests the difference is less than 1%.*

For example:

```
-- zxc gas diff in % (calc/real) -------------- 0.9995905254822616 (len: 4234)
-- gas used (real) -------------- 332133
-- zxc total gas used (real) -------------- 280821
-- zxc gas diff (calc-real) -------------- -124
-- zxc gas diff in % (calc/real) -------------- 0.999558437581235 (len: 4234)
-- gas used (real) -------------- 280821
-- zxc total gas used (real) -------------- 357821
-- zxc gas diff (calc-real) -------------- 443
-- zxc gas diff in % (calc/real) -------------- 1.001238049192194 (len: 4874)
-- gas used (real) -------------- 357821
-- zxc total gas used (real) -------------- 357826
-- zxc gas diff (calc-real) -------------- 47
-- zxc gas diff in % (calc/real) -------------- 1.0001313487560994 (len: 4810)
-- gas used (real) -------------- 357826
-- zxc total gas used (real) -------------- 428323
-- zxc gas diff (calc-real) -------------- 989
-- zxc gas diff in % (calc/real) -------------- 1.002309005119968 (len: 6090)
-- gas used (real) -------------- 428323
-- zxc total gas used (real) -------------- 425820
-- zxc gas diff (calc-real) -------------- 893
-- zxc gas diff in % (calc/real) -------------- 1.0020971302428257 (len: 6026)
-- gas used (real) -------------- 425820
-- zxc total gas used (real) -------------- 356928
-- zxc gas diff (calc-real) -------------- -254
-- zxc gas diff in % (calc/real) -------------- 0.9992883718845257 (len: 4554)
-- gas used (real) -------------- 356928
-- zxc total gas used (real) -------------- 305628
-- zxc gas diff (calc-real) -------------- -254
-- zxc gas diff in % (calc/real) -------------- 0.9991689243132174 (len: 4554)
-- gas used (real) -------------- 305628
```

We also added a service to our server that monitors real life transactions, to make sure our estimations stay accurate.

This analysis was performed by Pessimistic:

Daria Korepanova, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Evgeny Bokarev, Junior Security Engineer
Konstantin Zherebtsov, Business Development Lead
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

October 2, 2024