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

Kirobo FCT Security Analysis

by Pessimistic

This report is public

July 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 Intentable smart contracts. We described the audit process in the section below.

This report starts from the recheck #7. The initial audit and the previous rechecks are in the unpublished **"Kirobo FCT Security Analysis.pdf"** report. All issues from the last report were fixed or commented on, except one low severity issue.

The codebase was updated, and the recheck #7 showed several issues of medium severity: Documentation, Incorrect check of addresses, Insufficient code coverage, Unaccounted L1 gas, Unchecked size of calldata. Also, several low severity issues were found.

All tests passed. The project documentation did not include the description of the current diff.

After the recheck #7, the codebase was updated again.

The developers fixed the following issues of medium severity: Incorrect check of addresses, Unaccounted L1 gas, and Unchecked size of calldata. They commented on the Insufficient code coverage issue of medium severity, addressed the Insufficient documentation issue of medium severity, fixed all low severity issues and found new bug of medium severity about solver's signature. Also, they updated the documentation.

All the tests passed. Their number and the code coverage increased.

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.

General recommendations

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

Project overview

Codebase update #7

For the recheck, we were provided with Intentable project on a private GitHub repository, commit 7a65bc53412d4e6ef070ee9fcfda5823d6fee2b6.

The scope of the audit included diff between the previous and current commit and the new contracts:

- KIRO.sol;
- extensions/FCT_Ext_SignersValidator.sol.

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

We found several medium severity and low severity issues. 460 tests passed, and 24 tests had pending status. The code coverage was 82.84%.

Codebase update #8

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

The **Arch.sol** contract was split into three contracts, which were included to the scope:

- Arch_Arbitrum.sol;
- Arch_Mainnet.sol;
- Arch_Optimism.sol.

The developers fixed or commented on all the issues of medium severity.

All 592 tests passed. The code coverage was 84.64%.

Audit process

We started the audit on May 14 and finished on May 28, 2024.

It was the project's 7th recheck. The previous ones are available in the unpublished **"Kirobo FCT Security Analysis.pdf"** report. All issues from the last report were fixed or commented on, except one low severity issue.

We inspected the materials provided for the audit and contacted the developers for an introduction to the project.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

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:

- Whether the functionality with magic hash is safe;
- Whether it is not possible to charge more fees from users;
- Whether the updated code does not break the previous logic;
- Whether the gas in Optimism is calculated correctly;
- Whether the new functionality with signers validator works appropriately.

We scanned the project with Semgrep rules for smart contracts and sent the results to the developers in a text file.

We asked the developers to run tests and calculate the code coverage as the code could not be compiled.

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

We made the recheck on June 11-20, 2024. We checked whether the previous issues were fixed, read the updated documentation, and researched the developers' found bug, which could affect the solvers by passing any calldata.

We asked the developers to run tests and calculate the code coverage.

Finally, we updated 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 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.

M01. Insufficient documentation (addressed)

The project has documentation for the previous code version; however, the diff is not described there.

The full documentation is a critical part that helps to improve security and reduce risks. It should explicitly explain the purpose and behavior of the contracts, their interactions, and key design choices. It is also essential for any further integrations.

> *The developers added new sections describing the solvers' functionality.*

M02. Incorrect check of addresses (fixed)

It is supposed to be `l == j || varLocals.callFrom != mcalls.mcall[l].from` at line 578 in the `batchMultiSigCall` function of the **FCT_BatchMultiSig** contract. Otherwise, it will always revert when `varLocals.callFrom == call.from & call.functionSignature == MAGIC_HASH` at line 575.

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

M03. Insufficient code coverage (commented)

The Optimism version, included in the scope, is commented out in the **Arch** contract, so the tests did not cover it.

We always note the availability of tests and code coverage. We highly recommend covering this part of the code with tests.

> *Pessimistic's comment:*
> *The overall code coverage is good and is equal to 84.64%.*

> *Comment from the developers:*
>
> *The **Arch.sol** library is responsible for calculating extra common gas and extra gas. Due to the way L2s (Optimism, Base, and Arbitrum) operate and calculate transaction fees, we need custom logic in **Arch.sol** for each of the Layer 2 chains. This requires calling external contracts for additional calculations:*
>
> - *On Arbitrum - ArbGasInfo.sol;*
> - *On Optimism/Base - GasPriceOracle.sol.*
>
> *To test the calculations of **Arch.sol** libraries on L2s, we run the tests via Tenderly's DevNets. DevNet is a real-time fork, which essentially is a copy and a realistic environment of an L2 network, which allows us to verify if **Arch.sol** is returning accurate values.*

M04. Unaccounted L1 gas (fixed)

L1 gas is not added to the gas estimation in the `estimateExtraGas` function in the Optimism version of the **Arch** contract. It can lead to incorrect calculations of consumed gas and fees.

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

M05. Unchecked size of calldata (fixed)

The activator can add bytes to `calldata` after the expected `calldata`, when calling the `activate` function of the **FCT_Actuator** contract. It can increase the estimated gas in the `Arch.estimateExtraCommonGas` and `Arch.estimateExtraGas` as the value also depends on the calldata size. This issue is also applicable to the `activateBatch` function.

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. Identical contracts (fixed)

The **Oracle** and **RecoveryOracle** contracts are almost identical. Consider removing one of them or making the difference clearer to improve code readability.

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

L02. Unused variable (fixed)

The `owner` storage variable is not used at line 11 of the **Oracle** contract. Consider removing this variable.

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

L03. Unused import (fixed)

In the **FCT_Ext_SecureStorage**, and **FCT_Lib_UniswapV2** contracts, there is a redundant import and inheritance of the AccessControl library.

The issue has been fixed on commit 78c90bb9bf2b38c50b03ef3d89f0d125e2bc3d6a.

Notes

N01. Redundant getter (fixed)

The `RecoveryOracle.activator` and `Oracle.activator` functions return the `ACTIVATOR` variable, which has `public` visibility and can be read without a getter. Consider removing the `activator` function and renaming the `ACTIVATOR` variable to `activator` to match the interface.

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

N02. The creator is equal to zero address (commented)

The `creator` method always returns `address(0)` in the **StorageBase** contract. It has the comment "needed to pass compilation". However, this method is used in the contracts that are inherited from the `StorageBase` contract. Consider fixing it or making it `virtual` to allow overriding.

> *The `StorageBase.creator` reverts now. We have not checked the lll code of the minimal proxy (it is out of scope).*

> *Comment from the developers:*
>
> *`creator` function always returns the factory address. When the factory creates a new wallet (**SmartWallet** for example) it creates a special proxy, written in lll, that hijacks the creator function call and always returns the factory address.*
>
> *This makes the **SmartWallet** immune to bad updates, such as implementing creator function or overriding owner address or proxy's target data storage.*
>
> *The factory handles owner and version changes and has functions to recover the owner or version in such cases. The creator function is needed to pass compilation, so it can be used on from a solidity code (although it is hijacked by the lll code).*
>
> *It was updated to always revert instead of returning `address(0)` in order to make sure it is never being called directly.*

N03. Undocumented logic (fixed)

The value of the gas limit inside the call is not clear in the case of `MAGIC_HASH` in the **FCT_BatchMultiSig** contract, as the documentation does not provide a description.

And it is not obvious who pays the commissions and for gas if the activator adds an extra challenge or the solver wastes more gas (when using a magic hash).

> *The `MAGIC_HASH` constant has been renamed to `SOLVER_HASH`.*

> *The developers added the following comment to the documentation:*
> *It is recommended that all payers' indexes will be 0, except the solvers one (optional). In this way the solver will pay for the whole FCT, eliminating the common gas payment that might increase due to a long call message used by the solver.*

N04. 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.

> *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 N05 comment for more info about the need to pre-calculate some gas costs).*

N05. 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.

N06. Unassigned admin role (fixed)

The `DEFAULT_ADMIN_ROLE` is not initialized, and there is no reassignment `adminRole` of `X_ADMIN_ROLE` roles. This applies to the following contracts and roles:

- **FCT_Funding**: `MANAGER_ADMIN_ROLE`;
- **FCT_Tokenomics**: `DAO_ADMIN_ROLE`;
- **FCT_FlashLoan_Aave_v2**: `DAO_ADMIN_ROLE`;
- **FCT_Authenticator**: `MANAGER_ADMIN_ROLE`;
- **FCT_Controller**: `ACTUATOR_ADMIN_ROLE`, `LOCAL_ENS_ADMIN_ROLE`, `ENS_ADMIN_ROLE`, `TARGET_ADMIN_ROLE`;
- **FCT_ENS**: `LOCAL_ENS_ADMIN_ROLE`, `ENS_ADMIN_ROLE`;

These roles are assigned to the deployer (`msg.sender`), meaning only the deployer will have these roles throughout the contract's lifecycle. Probably, this is the expected behavior if the deployer is a multisig or DAO.

The issue has been fixed on commit 78c90bb9bf2b38c50b03ef3d89f0d125e2bc3d6a.

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

July 2, 2024