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

Kirobo FCT Security Analysis

by Pessimistic

This report is public

July 4, 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

Earlier we audited the previous version of the project.

In this report, we considered the security of Intentable smart contracts. We described the audit process in the section below.

The audit showed one issue of critical severity: The engine function can be called directly. We also found several issues of medium severity:
Incomplete update of the tokenomics address, Incorrect update of balance,
The discrepancy with documentation, Calling the wrong function. Also, several low severity issues and notes were found. All tests passed and the code coverage was insufficient.

After the initial audit, the codebase was updated. The developers performed several updates and added one new contract.

The developers fixed one critical issue, The engine function can be called directly, and the following issues of medium severity: Incomplete update of the tokenomics address,
Incorrect update of balance, The discrepancy with documentation and
Calling the wrong function . They also fixed low severity issues, fixed and commented notes.

After the discussion with the developers, we moved Replay attack issue to the notes section (previously it was C01). Also, we removed M02, M04, M07, and M08 from the report as false positives.

In addition, the developers found and fixed the following issues by themselves:
No data validation and No runner authentication.

The number of tests and the code coverage of the whole project decreased. Not all of the tests passed.

After the first recheck, the codebase was updated again. The developers updated some parts of the code from the previous scope. In addition, two new contracts were added to the scope.

We found the ERC20 standard violation issue of medium severity, several issues of low severity, and several notes. All tests for the current scope passed.

After the second recheck, the codebase was updated again. The update included fixes and the small change in functionality.

The developers fixed the ERC20 standard violation issue of medium severity. Also, they fixed or provided comments for all low severity issues and notes.

The number of tests increased. All tests passed.

After the third recheck, the codebase was updated again. The update included a new contract in the scope, and integration with Arbitrum and Uniswap V3.

The audit showed two issues of medium severity: Price oracle bug and Incorrect updating of balances in Kiro tokens.

The number of tests increased. However, the new tests were related to another scope (DAO). Not all tests passed.

After the fourth recheck, the codebase was updated again. The developers provided fixes for medium issues from the previous update and introduced new functionality, including a new contract **FCT_Token** and a library **Arch**.

The audit showed three new issues of medium severity, namely Unhandled ERC20 return values, Contradictory conditions, and Incorrect comparison. We also discovered two new low-severity issues.

The number of tests increased, but not all test cases pass.

After the previous recheck, the codebase was updated again. The developers fixed or provided the comment for the following medium severity issues: Unhandled ERC20 return values, Contradictory conditions, Incorrect comparison. Also, they fixed all low severity issues.

During the recheck, we added new item to the Unhandled ERC20 return values issue, found the new medium severity issue "Inaccurate accounting of tokens and ether" and one low severity issue.

After the recheck #6, we removed "Inaccurate accounting of tokens and ether" issue of medium severity as false positive.

The project has a complicated code structure and huge size functions. It may lead to bugs that are unrealistic to find.

General recommendations

We recommend fixing the rest of the issues, performing cosmetic codebase refactoring to enhance modularity and improve code readability. We also recommend implementing CI to run tests, calculate code coverage.

Project overview

Project description

Previously we made this audit at the commits b4ad34628272b5db09e5c4cda13f4122cf07c1d5 and f0f9e4ed3893577abc47ee8fd776229138126428.

For the current update, we were provided with Intentable project on a public GitHub repository, commit 857dd316a9373f96dea60e927bb04d050652828a.

The scope of this update included only the following files:

- economy/FCT_Actuator.sol;
- economy/FCT_ActuatorCore.sol;
- economy/FCT_ActuatorStorage.sol;
- economy/FCT_Funding.sol;
- economy/FCT_Tokenomics.sol;
- extensions/Decimals.sol;
- extensions/FCT_Ext_TokensSafeMath.sol;
- extensions/FCT_Ext_TokensValidator.sol;
- interfaces/IFCT_Actuator.sol;
- interfaces/IFCT_Controller.sol;
- interfaces/IFCT_Engine.sol;
- interfaces/IFCT_ENS.sol;
- interfaces/IFCT_Funding.sol;
- interfaces/IFCT_Runner.sol;
- interfaces/IFCT_RunnerOptional.sol;
- interfaces/IFCT_Tokenomics.sol;
- libraries/FCT_Lib_UniswapV2.sol;
- libraries/UniswapV2Library.sol;
- FCT_BatchMultiSig.sol;
- FCT_Constants.sol;
- FCT_Controller.sol;
- FCT_ENS.sol;
- FCT_Helpers.sol;
- RecoveryWallet.sol;
- uniswap-oracle folder.

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

280 tests out of 304 passed successfully, and 24 tests had pending status. The code coverage for the all project was 64%.

Codebase update #1

After the initial audit, the codebase was updated. We were provided with the commit ba4376c8e2536f0751b7ca9594de31a06ecdfde0. This update contained fixes and comments for all issues, and one new contract **FCT_Authenticator** which was included to the scope.

This version of the code had fixes for the issues that were found by the developers.

In this update, the number of tests of the whole project decreased to 209. 180 out of the tests passed, 24 had pending status, and 5 failed. The code coverage of the whole project was 38.3%.

Codebase update #2

After the recheck #1, the codebase was updated, and we were provided with commit d01b2b6237816c21aafde1dd324d89377e6b3d0b.

Two new contracts were added to the scope:

- FCT_Ext_SecureStorage.sol;
- FCT_FlashLoan_Aave_v2.sol.

We found one issue of medium severity, several of low severity, and several notes. The developers also made a small update to the code from the previous scope.

The number of tests for the FCT part of the project was 140. 139 out of the tests passed, and 1 test had pending status. The code coverage for these two contracts was 81.16%

Codebase update #3

After the recheck #2, the codebase was updated, and we were provided with commit 7bd40190a2479011b6db08a6ec058cbf4ded7760.

The developers fixed one medium severity issue, almost all issues of low severity and notes. For the rest of the issues they provided comments.

The number of tests for the FCT part of the project increased to 143. 142 out of the tests passed and 1 test had pending status. However, 1 test sometimes failed due to timeout. The code coverage for the FCT part of the project was 63.7%.

Codebase update #4

After the recheck #3, the codebase was updated, and we were provided with commit 8e995867c76b6b2669b0fa67315edfad8a3e0a5b.

The developers added **FactoryProxy.sol** file to the scope. We found 2 new issues of medium severity.

The number of tests increased to 171. 168 out of the tests passed (27 tests were not related to this scope) and 3 tests failed (2 tests were not related to this scope and 1 test sometimes failed due to timeout). The code coverage failed to run.

Codebase update #5

After the recheck #4, the codebase was updated, and we were provided with commit c927bca647875dd765f49d8eedc13032184791ac.

The developers added two new files to the scope: **FCT_Token.sol** and **Arch.sol**.

The number of tests increased to 192. 190/192 tests successfully pass, 1 test had pending status and 1 test failed. The code coverage stands at 48.8%.

Codebase update #6

After the recheck #5, the codebase was updated, and we were provided with commit [430c6782b079f9339b0a32eb5e82c8bc23606c2f](430c6782b079f9339b0a32eb5e82c8bc23606c2f).

The developers included two new files to the scope: **FCT_Funding.sol** and **FCT_Lib_MultiCall.sol** . They also fixed medium and low severity issues. However, we found one medium severity and one low severity issues.

The number of tests was 187. 172 out of the tests passed and one had pending status. The code coverage was 49,38%.

Audit process

We started the audit on January 23 and finished on February 3, 2023.

This is the second recheck of this project.

In this update, we inspected the materials provided for the audit. Then, we 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 the difference between previous commits, the new features and the new contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Whether replay attacks are impossible;
- Whether it is not possible to remove the contract through the `delegatecall`;
- Whether the code corresponds to the documentation;
- Whether the storage of the **FCT_ActuatorCore** and **FCT_Actuator** contracts is not broken;
- Whether it is not possible to purge FCT of other users;
- Whether the decoding of fix-sized arrays is correct;
- Whether the decoding of addresses and values is correct;
- Whether it is not possible to manipulate with a gas price;
- Whether changeable addresses can be changed in all parts of the code where they are used.

We scanned the project with the static analyzer Slither and our private plugin with extended set of rules and then manually verified their output.

We ran tests and calculated the code coverage.

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

After the initial audit, we discussed the results with the developers. On February 19, the developers provided us with an updated version of the code. In this update, they fixed most of the issues from our report. Also, a new **FCT_Authenticator** contract has been added to the scope.

We reviewed the updated codebase. We scanned the project with the static analyzer Slither and our private plugin with extended set of rules and then manually verified their output. We did not find any issues in the newly implemented functionality.

Finally, we updated the report.

From the 27th of February to the 2nd of March, we made the next recheck. This recheck included fixes for the previous scope and two new contracts.

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

We reviewed the updated codebase. We scanned the project with the static analyzer Slither and our private plugin with an extended set of rules and then manually verified their output.

We manually analyzed the difference between previous commits, and the new contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Whether the integration with Aave v2 is correct;
- Where the data in the **FCT_Ext_SecureStorage** contract cannot be replaced;
- How leftovers can be withdrawn after flashloan;
- Whether the flashloan logic works well with all ERC20 tokens;
- Whether there are checks of caller inside of `fctCall` functions.

Finally, we updated the report.

After that, from the 15th to the 16th of March we checked fixes for the scope and functionality update.

We reviewed the updated codebase. We scanned the project with the static analyzer Slither and our private plugin with an extended set of rules and then manually verified their output.

Finally, we updated the report.

On March 21 we added the latest developers comment and updated the report.

From the 7th to 13th June we checked the new contract from the scope and new integrations with Arbitrum and Uniswap V3. Then we updated the report.

During the period of June 29th to July 3rd, we reviewed the new contracts and their integration, including the **FCT_Token** which replaced the Kiro token for internal payments. Furthermore, we examined the functionality of validations and computed operations within the **FCT_BatchMultisig** contract.

Then we reflected the results in the report.

From the 24th to 26th July we made a recheck of the previous scope and checked two new contracts: how they integrated with the project and their logic.

All the fixed issues and which we found written up in the report.

On January 16, 2024, we rechecked issue M16 and marked it as false positive.

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.

C02. The engine function can be called directly (fixed)

It seems that there is no check who calls `batchMultiSigCall` function of the **FCT_BatchMultiSig** contract. It can be called directly. It allows to not pay fees to the builder, activator and the protocol.

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

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. Incomplete update of the tokenomics address (fixed)

The **RecoveryWallet** contract has the unchangeable address of the tokenomics. At the same time, the **FCT_ActuatorCore** contract can change versions of the tokenomics for the given id (the `setTokenomics` function at lines 67-78).

During the activation, the `fctPaymentApproval` function of the **RecoveryWallet** contract is called to check whether a wallet supports the address of tokenomics at line 291.

As a result, if the address of tokenomics for the given id is changed then a wallet will not support the new address. And the activation will not be allowed.

Consider adding functionality to update the tokenomics address in the **RecoveryWallet**.

> *Comment from the developers:* *Tokenomics is now connected to a pacific version (majur, minur, patch) of an engine.*

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

M03. Incorrect update of balance (fixed)

There is an incorrect variable in the contract **FCT_ActuatorCore** at lines 251-252. It would be correct as follows:

```
balance.kiro += kiro;
```

```
balance.eth += eth;
```

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

M05. The discrepancy with documentation (fixed)

In the **FCT_Constants** contract at line 122, the `VAR_ADDRESS_ACTIVATOR` is equal to `0x00FA0D0000000000000000000000000000000000`. However, according to the documentation, it should be equal to `0xFA0B0000000000000000000000000000000000`.

> *Comment from the developers:* *Changed from 0xFA0B0…00 to 0xFA0D00…00.*

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

M06. Calling the wrong function (fixed)

In `activateBatch` function of the **FCT_Actuator** contract `activateForFree` function is called inside the for loop at line 212. Nevertheless, according to logic, it should be the `activate` function.

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

M09. No data validation (fixed)

In the contract **FCT_BatchMultiSig**, in case of activation without a payer (the activator pays a commission), no verification is made that the EIP712 msg equal to the actual data that was signed.

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

M10. ERC20 standard violation (fixed)

ERC-20 standard states: "Callers MUST handle `false` from `returns (bool success)`. Callers MUST NOT assume that `false` is never returned!"

However, returned value from `approve` call is not checked at line 123 in the `executeOperation` function of the **FCT_FlashLoan_Aave_v2** contract.

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

M11. Price oracle bug (fixed)

The **FCT_Actuator** contract estimates the amount of ether for Kiro tokens based on the Uniswap V3 price oracle inside the `getAmountOfEthForGivenKiro()` function.

However, the function mistakenly returns the amount of Kiro for given ether, because `estimateAmountOut()` is being called at lines 577 and 579 with `tokenIn=ether` and `tokenOut=KIRO`.

The `getAmountOfKiroForGivenEth()` function contains a similar issue at lines 536 and 538.

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

M12. Incorrect updating of balances in Kiro tokens (fixed)

During distribution of ether between builder and activatior, the same proportion of Kiro tokens should be subtracted from their balances in the `activate` function of the **FCT_Actuator** contract. Now only activator's kiro tokens are decreased at line 394.

This can lead to the builder having more ether and Kiro tokens than they should have.

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

M13. Unhandled ERC20 return values (commented)

ERC-20 standard states: “Callers MUST handle false from returns (bool success). Callers MUST NOT assume that false is never returned!” However, the code in the **FCT_Token** file does not consistently check the returned values from `approve, transfer`, and `transferFrom` calls.

The same case has been found in the `erc20Approvals` function of the **FCT_Lib_MultiCall** contract on the commit `430c6782b079f9339b0a32eb5e82c8bc23606c2f`.

> *Comment from the developers:*
>
> *The mentioned calls can never return false, so there is no need to implement error handling. Adding error handling would only result in unnecessary gas costs.*

M14. Contradictory conditions (fixed)

The `_validate` function of the **FCT_BatchMultiSig** contract has two contradictory conditions and lines 983 and 984. We recommend removing the `if-case` and using the existing `require` statement. This updated implementation will enable calling the `_validate` function recursively to handle another validation as a value.

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

M15. Incorrect comparison (fixed)

In the `_onlyPrevComputed` function of the **FCT_BatchMultiSig** contract, at line 1301, the condition `(variable & VAR_MASK) < compIndex)` should be modified to `(variable & VAR_MASK) >= compIndex)`. This change aligns with the logical requirement of using only previous results in calculations. A similar case can be found in the `_validate` function at line 984.

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. Lack of the second token check (fixed)

There is the check that one of the tokens is Kiro token at line 65 of the **FCT_Actuator** contract. And the other token does not have the check that it is a token equated to ETH.

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

L02. Unnecessary initialization of variables (fixed)

The contract **FCT_ActuatorCore** is designed to execute logic in the contract **FCT_Actuator** by calling the function `delegatecall`. In this case, the **FCT_Actuator** storage is used for both contracts.

That is why it is not necessary to initialize storage variables in the **FCT_ActuatorCore** contract.

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

L03. Different values of variables (fixed)

The **FCT_ActuatorCore** contract is an extension of the **FCT_Actuator** contract and should use the same values of `immutable` variables.

Consider adding the check in the constructor of the **FCT_Actuator** contract that both contracts have the same `immutable` variable values.

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

L04. Dependency management (fixed)

- The OpenZeppelin version used is not specified in the `package.json`;
- There is no `package-lock.json` or `yarn.lock` files.

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

L05. Double check (fixed)

In the `_chargeActivator` function of the **FCT_Actuator** contract the `kiroboShare` limit check looks redundant at line 523 since the same check is inside of the **FCT_Tokenomics** contract at line 90. This could be done with the expectation that there may be several activators, and they will have their own limitations.

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

L06. Gas consumption (fixed)

To reduce gas consumption, consider setting a local variable with `s_staked[activator]` value. Since there are two readings from the storage and multiple calculations of keccak256 at lines 239–242 in the `activate` function of the **FCT_Actuator** contract.

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

L07. Misleading comments (fixed)

Comments do not correspond to variables at lines 46–52 in the **FCT_BatchMultiSig** contract.

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

L08. No runner authentication (fixed)

There is no caller verification in the **FCT_BatchMultiSig** contract in the `batchMultiSigCall` function. Anyone can copy the vault contract and run FCT on its behalf. As a solution, consider adding contract authentication.

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

L09. Implicit visibility (fixed)

Storage variables are considered internal by default. Consider declaring visibility of the variables explicitly to improve code readability in the following cases:

- at lines 30-31 of the **FCT_FlashLoan_Aave_v2** contract;
- at lines 18-24 of the **FCT_Ext_SecureStorage** contract.

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

L10. Unused variable (fixed)

The following storage variables are not used in the code:

- `s_forFreeFeesLimitBPS` at line 34 from the **FCT_ActuatorStorage** contract;
- `fct_bytes32`, `fct_uint256`, `fct_bytesat` at lines 22-24 from the **FCT_Ext_SecureStorage** contract.

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

L11. Unused event (fixed)

The `FCTE_FctBytes32Updated` event is declared at line 54 in the **FCT_Ext_SecureStorage** contract but is not used in the code. It seems that it should be called in the `_write_fct_bytes32` function instead of `FCTE_PrivateBytes32Updated` event.

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

L12. The return value is not used (commented)

The `activate` function from the **FCT_Actuator** contract returns a value, but it is not checked and is not used in the `executeOperation` function of the **FCT_FlashLoan_Aave_v2** contract at line 132.

> *Comment from the developers: The return value from the activate function is not necessary and only there to save a read to the activator that runs the trx, there is no use for this in the flashloan contract.*

L13. Typo in the structure field (fixed)

The field of the `FLAmount` structure at line 19 in the **FCT_FlashLoan_Aave_v2** contract has a typo and should be called `premium` instead of `remium`.

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

L14. Implicit constant visibility (fixed)

Storage variables are considered internal by default. Declare visibility of `PRIVATE_STORAGE_HASH` and `FCT_STORAGE_HASH` variables at lines 11-12 of the **FCT_Ext_SecureStorage** contract explicitly to improve code readability.

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

L15. Improper hooks usage (fixed)

According to the OpenZeppelin documentation, the `_beforeTokenTransfer` function should be declared as virtual. However, in the **FCT_Token** contract, it is not currently marked as virtual.

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

L16. Version check mismatch (fixed)

In the `fctIsVersionSupported` function of the **RecoveryWallet** contract, consider using `MAJOR_VERSION_PREV` in one of the comparisons on lines 258 and 259 to ensure backward compatibility with the previous version. Currently, these two comparisons are identical and use `MAJOR_VERSION`.

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

L17. Incorrect comment

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

Notes

N01. Not finished code (commented)

Some of the functionality has not been finalized yet. For instance:

- The `activation.executing` field is not used yet;
- According to the comment at lines 21-22, the **FCT_Funding** will be further developed. It is unclear where the money for this contract comes from, since users send ETH to the **FCT_Actuator** contract;
- The **FCT_Controller** can support multiple activators because of `ACTUATOR_ROLE`. At the same time, the **FCT_Funding** contract has `immutable` variable `s_actuator`. It means that this contract can support only one activator.

> *Comment from the developers:*
>
> *`activation.executing` is used in the **SecureStorage** contract throw `isExecuting` function.*
>
> *The **FCT_Funding** contract will be able to swap ETH it receives from **FCT_Actuator** to kiro, the implementation will be added later (swap from uniswap, or kiro tokens Kirobo will add to it…) this part is still under development.*
>
> *The controller by design supports multiple actuators (3rd party will want to have their own actuator). But the **FCT_Funding** also by design supports only specific actuator for each funding contract.*

N02. Magic numbers (fixed)

Consider declaring the following letters as constants:

- In the contract **FCT_Tokenomics** at lines 88-96, 270-271;
- In the contract **FCT_BatchMultiSig** at lines 325, 478-479, 485, 563, 611;
- In the contract **RecoveryWallet** at lines 248, 250;
- In the contract **FCT_ENS** at line 110;
- In the contract **FCT_Controller** at line 361.

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

N03. Hardcoded gas cost (commented)

The **FCT_Tokenomics** contract uses pre-calculated costs for gas at lines 239, 246-247. If execution costs of opcodes change with the new Ethereum hardforks, the contract will require re-deployment.

> *Comment from the developers: FCT_Tokenomics.sol contract is upgradable, we know that the calc is relevant for the current fork and by upgrading we will be able to update the hard-coded numbers.*

N04. Very long functions (commented)

There are too long functions in the project:

- `abiToEIP712` (at lines 738-861) and `batchMultiSigCall` (at lines 203-721) functions from the **FCT_BatchMultiSig** contract;
- `activate` function (at lines 228-495) from the **FCT_Actuator** contract.

Consider splitting these functions 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.*

N05. Unsafe delegatecall using (commented)

The using of `delegatecall` is potentially unsafe since the `target` contract may call `selfdestruct`.

In the `fctCall` function of the **RecoveryWallet** contract, `delegatecall` can be called by `ENS_LIBRARY` type target only. And targets of this type can be added through `FCT_ENS.setLocalEns` function by `LOCAL_ENS_ROLE`.

So, this `delegatecall` is protected by the project role. However, if admin's private keys become compromised, any targets can be added.

> *Comment from the developers: In the function `ensToAddress` the second param can be an address or 0, for extra security the user can choose to add the lib address, and so, if the ENS of the lib was changed the FCT will not work.*

N06. Overpowered roles (commented)

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

- **FCT_Actuator**, **FCT_ActuatorCore** contracts:
  - `PROTECTOR_ROLE` can update the price of Kiro token;
  - `DAO_ROLE` can set the addresses of **FCT_Tokenomics**, **FCT_Funding** contracts, freeze and pause the ability to pay in ETH, set the value of ETH penalty and set other parameters and limits for FCT activation;
  - `MANAGER_ROLE` can set builder, the project wallet address;
- `DAO_ROLE` of **FCT_Tokenomics** contract can:
  - enable/disable free mode for FCT payers;
  - change parameters that influence payments and gas cost refunds;
- **FCT_ENS** contract:
  - `ENS_ROLE` can change the address of the **ENS** contract, which is used for adding new targets;
  - `LOCAL_ENS_ROLE` can add new targets. This role has an impact on the `delegatecall` in the **RecoveryWallet** contract (see the Unsafe delegatecall using issue);
- **FCT_Controller** contract:
  - `TARGET_ROLE` can add a new engine. This has an impact on the whole system since the wrong address can lead to unpredictable consequences;
  - `ACTUATOR_ROLE` can call the engine through the `fallback` function;
  - `LOCAL_ENS_ROLE` (is not used anymore);
  - `ENS_ROLE` can set the address of the **FCT_ENS** contract.

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.*

N07. Replay attack (commented)

FCT can be executed multiple times, even more, than expected in the `repeatsLeft` field of `s_fcts` mapping from the **FCT_Controller** contract.

It can happen since FCT can be purged by anyone and then registered again:

- The `purge` function of the **FCT_Controller** contract deletes `s_fcts[messageHashes[i]]` variable at line 192;
- There is no check that it can be removed only by the owner of FCT. That is why anyone can delete any FCT with a true purgeable flag.

> *Comment from the developers:* *FCT can be purged only when expire time has passed and it cannot be executed anymore.*

N08. Non-zero balance after flashloan (commented)

Any leftovers in the **FCT_FlashLoan_Aave_v2** contract balance after flashloan can be withdrawn by anyone via FCT to any address since `target` at line 82 can be any address.

> *Comment from the developers:* *We decided not to tract the leftovers in the contact (saves a lot of hassle and gas) and if there are leftovers in the contract they can be withdrawn by any FCT, we are aware and prefer it like that in order to keep the contract simple and to not have locked funds in the contract.*

N09. Collisions (commented)

If the `keccak256` hash is calculated from the result of `abi.encodePacked(a, b)`, the collisions are possible since both the following cases will give the same hash: a=AAA, b=BB and a=AA, b=ABB.

The `write_uint256`, `write_bytes32` and `write_bytes` functions store data to the `private_uint256`, `private_bytes32` and `private_bytes` mappings in the **FCT_Ext_SecureStorage** contract. The data mappings can be changed in the `fctCall` function via the FCT running.

And if `abi.encodePaked(fctMsgHash, key)` == `abi.encodePacked(msg.sender, key)`, then the data can be replaced via the FCT. Theoretically, it is possible since `fctMsgHash` is a hash of FCT data and `key` is a free parameter.

> *Comment from the developers: In order to reduce the chance of collision to the level of the network itself we use the following:*
>
> 1. *There is a new function in FCT_Controller that verifies a given messageHash equals the current running FCT messageHash.*
> 2. *This function is used in SecureStorage to make sure that the messageHash is genuine.*
> 3. *Because the FCT holds also a random number (part of the message meta), theoretically the FCT messagehash can be manipulated to match an existing address. In order to eliminate this possibility there is a a constant hash, at the beginning of the slot data to be hashed, that indicates this slot belongs to a specific FCT (rather than private slots that belongs to Runners).*
> 4. *FCT slots: keccack256(<fct_hash(bytes32)> <fct_messageHash(bytes32)> <key(bytes32)>).*
> 5. *Runner slots: keccack256(<runner_hash(bytes32)><msg.sender(address)> <key(bytes32)>).*
> 6. *FCT_Controller also makes sure that only one FCT messageHash can be registered in the system (assuming purge flag is false).*

N10. USDT approve (fixed)

The approve function of USDT does not revert if the value you passed is 0 or if the `allowed` for the current `msg.sender` and spender is 0.

After each flashloan, the `executeOperation` function of the **FCT_FlashLoan_Aave_v2** contract can be temporarily locked due to the revert of `approve` function at line 123.

It can be unlocked via the FCT with zero approval inside. However, this makes using the FCT more complicated.

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

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Daria Korepanova, Senior Security Engineer
Ivan Gladkikh, Security Engineer
Vladimir Pomogalov, Security Engineer
Nikita Kirillov, Security Researcher
Irina Vikhareva, Project Manager

July 4, 2024