EMBER FUND

Ember Security Analysis

by Pessimistic

This report is public.

Published: December 9, 2020

Abstract

In this report, we consider the security of the smart contracts of Ember Fund project. Our task is to find and describe security issues in smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. One 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, security audit is not an investment advice.

Summary

In this report, we considered the security of the smart contracts of Ember Fund project. We performed our audit according to the procedure described below.

The initial analysis showed two critical issues (hardcoded conversion rate and no replay protection), several issues of medium severity, including underflow, code logic, expensive loops, missing check if oracle provided actual data. Also, many issues of low severity were found.

After the audit, some fixes were applied, and the code base was updated to the latest version. All critical issues were fixed, also one issue of medium severity and several low-severity issues were fixed.

The project has no documentation.

General recommendations

We recommend fixing the rest of the issues and adding detailed documentation to the project. We also recommend using linters and static code analyzers as an essential part of development process.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether code logic corresponds to the specification.
2. Whether the code is secure.
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan project's code base with automated tools: Crytic, MythX, and SmartCheck.
  - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
  - We inspect the specification and check whether the logic of smart contracts is consistent with it.
  - We manually analyze code base for security vulnerabilities.
  - We assess overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

Project overview

Project description

For the analysis, we were provided with code base on private GitHub repository of Ember Fund project.

The scope of the audit included the changes made to `atomic-swaps` branch, commit a062d0c9f4ceb830978c96932238ba629aabe992, after the fork from `master` branch, commit 3e74a955c1e7359d6b16bd665f232c281ee7fb6c.

The project has no documentation.

Crytic and MythX cannot be used for the project as is. We have instructed the team on how to modify the project, so it is possible to introduce the tools into the development workflow.

Latest version of the code

After the audit, the code base was updated. For the recheck we were provided with commit 973befd36337282cfc3c4351deff713211f53349.

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 smart contracts security. We highly recommend fixing them.

Hardcoded conversion rate (fixed)

In **contracts/oracle/PriceOracle.sol**, there is the condition in `getPrice()` function:

```
if (stableCoins[_assetOne] == true && stableCoins[_assetTwo] == true) {
    return 100000000;
}
```

As a result, the conversion rate for both stablecoins will always be 10^8 in both directions.

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

No replay protection (fixed)

`addModule()` and `removeModule()` functions of **SmartWallet** contract do not process nonce. These functions do verify admin's signature; however, they allow sending the same data multiple times.

As a result, once admin used one of these functions, anyone can repeat the same call.

We recommend adding `_checkReplayAttack(_nonce)` check to protect against repeating calls.

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 current implementation. We highly recommend addressing them.

Underflow

In **modules/atomic_swaps/AtomicSwapERC20ToERC20Module.sol** at lines 177 and 181, there is a possible underflow if `decimals` value of ERC20 contract is larger than `18`:

```
uint256 normalizedQuote = quote.div(10 ** (18 - closeContractDecimals));
```

For such ERC20 contracts, it will be impossible to close an order.

Expensive loop

`getSwapIdsAsWatcher()` function of **AtomicSwapERC20ToERC20Module** contract has an expensive loop and thus will fail due to `out-of-gas` if an account is active enough.

Code logic (fixed)

In **AtomicSwapERC20ToERC20Module** contract, functions `getSwapIdsAsSmaAdmin()` and `getSwapIdsAsWatcher()` restrict access to on-chain data. However, all the data on blockchain including swap ids are publicly available.

Moreover, `getSwapIdsAsWatcher()` function leaks id count.

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

Outdated oracle response

Both **BtcOracle** and **EthOracle** contracts do not check if a response from the oracle is fresh enough.

Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

Project management

- There are Solidity files outside **contracts/** folder. Consider storing all Solidity files of the project in this folder.
- Contracts, configs, and dependencies of the project suggest inconsistent solidity versions. Consider updating `pragma version` in all files to `0.6.12` or later.
- Dependency management in **package.json** file is messed up: `truffle` is not a `devDependency`, OZ contracts are imported twice, etc.

Misleading messages/comments

- In **AtomicSwapERC20ToERC20Module** contract at line 76, should be `"Valid swap"` instead of `"Invalid swap"`.

  The issue has been fixed and is not present in the latest version of the code.
- In `getPrice()` function of **PriceOracle** at line 79, the comment states:

  ```
  1) Check to see if a direct or inverse oracle of the pair exists
  ```

  However, the code does not check for inverse oracle.

  The issue has been fixed and is not present in the latest version of the code.
- In **ETHOracle** contract, the comment at lines 11–12 was left from **BtcOracle** contract.

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

Code quality

- In `canValue()` function of **SetV1Valuer** contract, consider calling

  ```
  IRebalancingSetToken(_rebalancingSetToken).currentSet()
  ```

  or checking returned data with `abi.decode(returnData, (address))`.
- Avoid using hardcoded addresses as in **PriceOracle** contract at lines 70 and 72.

  The issue has been fixed and is not present in the latest version of the code.
- In `_getPrice()` function of **PriceOracle** contract at line 137, the use of `SafeCast` library is excessive. Consider performing a check `price >= 0` instead.
- **BtcOracle** and **EthOracle** contracts are almost the same. Consider storing only one instance and deploying it twice with different arguments. Hardcoded addresses should be replaced with constructor arguments.

Code style

- Both **BtcOracle** and **EthOracle** contracts implement **IOracle** interface implicitly. We recommend declaring it explicitly.

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

- “Magic” constants are used in the code all over the project. Consider assigning them names to improve code readability.
- Consider using `external` visibility level instead of `public` where possible.

Gas consumption

- Consider enabling optimization in **truffle-config.js**.
- Using `length` property of arrays in loops is expensive since it is read on each iteration. Consider using a local variable to optimize `.length` access.
- In **AtomicSwapERC20ToERC20Module** contract at line 199, consider making internal call instead of external.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst
Alexander Seleznev, Founder

December 9, 2020