Smart Contract Audit

Obolus DEX

EOS42

Report created on 27 May 2019

Executive Summary

The contract conforms to all standards and is in very good condition. There are no findings to report. The code follows all best practices for smart contract development on the EOSIO platform. There are no known vulnerabilities and no outstanding issues.

This table shows the severity matrix of all findings related to this smart contract audit.

Remark	Minor	Major	Critical
3	0	0	0

Auditing Strategy and Techniques Applied

EOS42 performed an initial review of the smart contract code as written and last updated on 17th May 2019. All of the main contract files were reviewed using the following tools and processes.

Throughout the review process, care was taken to ensure that the token contract:

- Documentation and code comments match logic and behavior
- Users keep control of their funds at all times
- Whoever deploys the contract does not have access to user funds
- Is not affected by any known vulnerabilities

Our team has followed best practices and industry-standard techniques to verify the proper implementation of the smart contract. Our smart contract developers reviewed the contract line by line, documenting any issues as they were discovered. Our strategies consist largely of manual collaboration
between multiple team members at each stage of the review, including:

1. Due diligence in assessing the overall code quality of the codebase.
2. Testing contract logic against common and uncommon attack vectors.
3. Thorough, manual review of the codebase, line-by-line.

Structure Analysis

The Obolus DEX smart contract consists of 4 files, 1 header file containing declarations of variables, and 3 source code files containing function definitions. The source code is split modularly into the main DEX component, the OTC market, and helper functions.
The code is then further split up into several functions and actions, with the functions performing work behind the scenes, and the actions being the interface to the users and to the smart contract itself.

Complete Analysis

General Comments

1. Remark Change all instances of eosio_assert to eosio::check, as assert is deprecated.
2. Remark Unless intending to change who actually pays for RAM on the system, use eosio::same_payer as the person to be billed for RAM when using multi_index::modify

helper.hpp

There are no findings for this file

obolusdex.cpp

1. Remark The obolusdex::fill function could cause a transaction timeout error if an order taker puts in such a large order that the cpu usage goes above 30ms whilst looping through the orders. This can also happen if order makers put in lots of very small volume orders. This can be fixed by creating a timeout condition on the loop, storing the state in an intermediate table, and having it continue at the next action that is called. Note: This is accounted for in the front end of the Obolus DEX, which calculates how many orders will be fulfilled and splits up larger orders into smaller orders before submitting them to the chain.

obolusdex.hpp

There are no findings for this file

otc.cpp

There are no findings for this file

Addressed Problems

1. Critical ✓ Addressed

   ACTION obolusdex::dplaceorder(const name user, const bool isBid, const asset baseVol, const asset quotePrice, const name baseAcc, const name quoteAcc, const uint64_t boardId)

   This action should require authorization of user. Otherwise someone else could place an order on behalf of a different user.
   Sending this transaction with ID 0 means that this can only happen once per block, multiple users could not call this actions in the same block as it would give a uniqueness constraint error. If a single user wanted to call this action multiple times, a nonce feature would need to be added along with the constantly changing send ID number.
2. Minor ✓ Addressed

   ACTION obolusdex::rmallobs()
   This function is very likely to time out if there are many boards with large orderbooks Consider that if the limit of 100 orders are deleted, that the function should call itself in a deferred transaction a few times, this will make the system recursively try to delete all entries as appropriate without risk of the system failing due to a timeout error
3. Remark ✓ Addressed

   inline int64_t obolusdex::fill(Order& takerOrder, obolusdex::orderbook_t& orderbook, const boards_t::const_iterator& boardIt)
   Consider using reverse iterators if the takerOrder is not a bid
4. Remark ✓ Addressed

   void obolusdex::printob(obolusdex::orderbook_t& orderbook)const
   This could time out if the orderbook is large
5. Remark ✓ Addressed

   ACTION obolusdex::rmtablebal(name scope, uint64_t nMax)
   Consider putting the n++ within the while loop to increase readability and reduce the risk for logical errors
6. Remark ✓ Addressed

   void bytearrayToHex(std::array<uint8_t, 32> hashdata, char* sss)
   This function implicitly expects the char* to be 64 bytes large.

All Contract Files

Filename	**SHA256**
helper.hpp	2d91f5cbac93574a23f3691d446ef5462b2d5e522cc53c3cbf6a6a1464f1777c
obolusdex.cpp	6ff9ea81d849a236efc9c69d105fe576a5e32209346e88e7c57fbcd79e41341f
obolusdex.hpp	388b565d3e07d4163477ea5fc3afb34a448eae1118c4f765ef2e1bcc2326eaa9
otc.cpp	99c78121263ba6c25a7ef9049a9782dc98b0b358466ce84fb554559425eb0f4f
obolus.wasm	63568444e2cb1709297334cfa51838e2afc41df5e5eb80db408d253688b1ffb1

I hereby confirm that the above audit is correct and true to the best of my knowledge and abilities.

[illegible]

....

Phillip Hamnett

EOS42