G0 Group

Security Review of
dFusion Exchange

January 27, 2020

Overview

G0 Group was engaged to perform a security review of Gnosis DFusion decentralised exchange smart contracts. G0 Group was contracted for a four person-week effort to that end. The primary subjects of this review were the smart contracts which implement Gnosis' dFusion Exchange: a fully decentralized trading protocol which facilitates ring trades via discrete auction between several ERC20 token pairs. This review was initially performed on https://github.com/gnosis/dex-contracts/commit/949f2e2f9d05a3570cb4dd5f4d1d54ecd5f5c009.

Files in Scope

```
contracts/
    libraries/
        TokenConservation.sol
    BatchExchange.sol
    DevDependencies.sol
    EpochTokenLocker.sol
@gnosis.pm/
    solidity-data-structures/
        contracts/
            libraries/
                IdToAddressBiMap.sol
                IterableAppendOnlySet.sol
    owl-token/
        contracts/
                TokenOWL.sol
```

Result Summary

During the course of this review, 4 security issues and 3 efficiency or usability issues were discovered, reported, and addressed.

No further issues were discovered in https://github.com/gnosis/dex-contracts/commit/6be551d8f0ec2ee0f0fe4c78301e4d303c4d57b6

Issues

1. Mix of an account address provided through a method argument through `msg.sander` allows bypassing of withdrawal lock

Type: *security* / ***Severity:*** *critical*

Function `withdraw` of `TokenConservation.sol` contains `msg.sender` in a check on `line 107` while the rest of the function works with `user` variable that is one of the function's arguments. This oversight allows a malicious attacker to bypass token withdraw lock that is critical for the correct functioning of the system.

Fix Description:

The issue was fixed in this commit:
https://github.com/gnosis/dex-contracts/pull/399/commits/256eaf71975b4d22b41d2a2bec517b6ed714c705

2. Balances that are prepared for withdrawal in the current batch can still be traded but are not considered in calculation of disregarded utility compromising the system of incentives

Type: *flawed incentives* / ***Severity:*** *medium*

While `subtractBalance` used on `line 351` of `BatchExchange.sol` can reduce the user's balance below the amount that he requested for withdrawal in current batch, `getBalance` on `line 706` considers this amount as if it was already withdrawn, leading to an inconsistency between the amount of tokens that the system thinks are available for the trade in it's disregarded utility calculation and actual available tokens. This might allow some users to "hide" the actual amount of tokens available for sale from the system leading to a better position in the context of market orders.

Fix Description:

The issue was addressed by preventing `subtractBalance` function from decreasing user's balance below the amount that is withdrawable. And is no longer present in the following commit:

https://github.com/gnosis/dex-contracts/commit/cd6dfb291262a890018498be05a58ab50e47d312

3. Disregarded utility penalty can be bypassed if users split their orders into multiple ones with smaller volume

Type: *flawed incentives /* ***Severity:*** *medium*

Traders can allow solvers to bypass penalty for solutions that don't use all tradable volume at provided prices by splitting their orders into multiple smaller ones since the system only considers unused volumes in orders that are part of the trade and not all orders available in the batch.

Client's response:

In theory, we would like to calculate the disregarded utility over all orders. Unfortunately, this is not in scope for the first MVP. We are aware that there are different methods in order to benefit from this imperfection. The mentioned strategy is not risk-free, as splitting orders also increases the risk of not being included in the solution. Solvers might exclude orders with a small volume, as the gas costs for including the orders might be higher than the reward for including them.

4. IdToAddressBiMap library has an overflow issue leading to unexpected behavior

Type: *unexpected behavior /* ***Severity:*** *major*

`IdToAddressBiMap.sol` has an overflow issue in the `insert` function that leads to malformed state when `uint16(-1)` is provided as `id` , this issue is not exploitable in the context of the audited contracts but might be in future uses.

Fix Description:

The issue was fixed and is no longer present in this commit:
https://github.com/gnosis/dex-contracts/commit/cd6dfb291262a890018498be05a58ab50e47d312

5. First return value of getTradedAmounts is just a copy of the first argument and doesn't seem necessary

Type: *efficiency* / ***Severity:*** *minor*

Client's response:

Yes, returning the buyAmounts is not necessary. We decided to stick with the current implementation, as it enhances the readability of the code

6. Passing an array to TokenConservation.init is unnecessary

Type: *efficiency* / ***Severity:*** *minor*

Client's response:

We decided to stick with the current implementation in order to keep the logic for the token conservation data creation encapsulated in TokenConservation library. The compiler undoes this implementation detail and our implementation does not cost more gas.

7. lastCreditBatchId is not reverted in undoCurrentSolution

Type: *usability* / ***Severity:*** *minor*

Client's response:

We decided against a reversion of the `lastCreditBatchId`, in case a better solution is submitted for the current batch. We believe that it is very unlikely that a user would benefit from this reversion. They would only benefit if they were intending to withdraw immediately tokens, which they traded in a previous solution of the current batch, but not in the most recently submitted solution. We prefer the simplicity of the contract over the very unlikely, unnecessary withdraw-blocks for a user.