Post-Audit Fix Report

Author: Nico Elzer

Commit: https://github.com/nicoelzer/gnosis-protocol-relayer/commit/319d6ff3ece08e2b854f0d7b0a8a0fe12893ba79

Critical Defects

Defect	Status
2.1 Incorrect Order ID in Cancellation	✅ Addressed

Moderate Defects

Defect	Status
3.1 Suboptimal Order Execution due to Known Issues	Not addressed due to general known issue with Gnosis Protocol, not specific to this relayer.
3.2 Desired Token Output Ignored	✅ Addressed
3.3 Possible Integer Truncation (1)	✅ Addressed
3.4 Incorrect Bounds Checks	✅ Addressed

Minor Defects

Defect	Status
4.1 Late State Change	✅ Addressed
4.2 Unnecessary Payable Modifier	✅ Addressed
4.3 Possible Integer Truncation (2)	✅ Addressed
4.4 Redundant Bounds Check	✅ Addressed

DXswap Gnosis Oracle Audit No. 1

Dec, 2020

Contents

Chapter 1

Introduction

1.1 Scope of Work

This code review was prepared by Sunfish Technology, LLC at the request of members of dxDAO, an organization governed by a smart contract on the Ethereum blockchain. The code covered by this review (see section 1.2) is functionality designed to allow dxDAO to trade DAO-managed tokens on a decentralized exchanged called Gnosis.

1.2 Source Files

This audit covers code from the public Github repository `https://github.com/nicoelzer/gnosis-protocol-relayer`.

Only code from the following git SHA was reviewed:

`0517a2c05d05797c30832fba75603339b7262e11`

Within that revision, only the following files received line-by-line review:

- contracts/GnosisProtocolRelayer.sol

This review was conducted under the optimistic assumption that all of the supporting software infrastructure necessary for the deployment and operation of the reviewed code works as intended. There may be critical defects in code outside of the scope of this review that could render deployed smart contracts inoperable or exploitable.

1.3 License and Disclaimer of Warranty

This source code review is not an endorsement of the code or its suitability for any legal/regulatory regime, and it is not intended as a definitive or exhaustive list of defects. This document is provided expressly for the benefit of dxDAO developers and only under the following terms:

Chapter 2

Critical Defects

Issues discussed in this section are defects that lead to the code to misbehave in ways that are directly exploitable and have severe consequences like loss of funds.

2.1 Incorrect Order ID in Cancellation

The code on line 170 trhough 172 of `GnosisProtocolRelayer.sol` uses the wrong ID for `IBatchExchange.cancelOrders()`:

```
uint16[] memory orderArray = new uint16[](1);
orderArray[0] = uint16(orderIndex);
IBatchExchange(batchExchange).cancelOrders(orderArray);
```

The code ought to use `order.gpOrderId` instead of `orderIndex`. This code works by coincidence when all orders are successfully placed with `IBatchExchange.placeOrder()`, since order IDs are implemented as indices in an array, much like `orderIndex` in the calling contract. However, this code will stop working once an order in `GnosisProtocolRelayer` expires without being placed, as `orderCount` will increase, but the number of orders in the `BatchExchange` contract will not.

Chapter 3

Moderate Defects

Issues discussed in this section are code defects that may lead to unintended deviations in behavior. It may be possible to chain multiple moderate defects into a working exploit.

3.1 Suboptimal Order Execution due to Known Issues

The Gnosis Protocol exchange has a number of known issues that impact the reliability of order execution.

The "Fake-token utility" issue described in the Gnosis API docs are of particular concern for this use-case, as it allows an attacker to force orders to receive worst-case execution costs. In practice this means an attacker can force every trade executed as a result of `placeTrade()` to only yield `expectedAmountMin` tokens, rather than a result closer to `expectedAmount`.

See https://docs.gnosis.io/protocol/docs/devguide03/#known-issues.

3.2 Desired Token Output Ignored

The `tokenOutAmount` field of the `Order` structure is unused by `placeTrade()`, even though it is stored into each `Order` by `orderTrade()`.

Either `placeTrade()` is computing `expectedAmountMin` incorrectly by not computing the output amount as a delta from `tokenAmountOut` rather than `expectedAmount`, or the functionality is vestigal and should be removed entirely.

(It seems as if `placeTrade()` should at least check that `expectedAmountMin` is within `priceTolerane` of `tokenAmountOut`.)

3.3 Possible Integer Truncation (1)

On line 158 of `GnosisProtocolRelayer.sol`, the expression `uint128(expectedAmountMin)` can yield unexpected results when the value of `expectedAmountMin` cannot be repre-

sented with 128 bits or fewer. Consider explicitly discarding values above `2**128`.

3.4 Incorrect Bounds Checks

Lines 132, 165, 167, 178, and 207 all use the following incorrect bounds check:

```
require(orderIndex <= orderCount, ...);
```

An `orderCount` of 1 implies that the only valid `orderIndex` value is zero. The correct bounds check is:

```
require(orderIndex < orderCount, ...);
```

Chapter 4

Minor Defects

Issues discussed in this section are subjective code defects that affect readability, reliability, or performance.

4.1 Late State Change

The `placeTrade()` function uses `order.executed` to ensure that trades are placed exactly once. However, the check for `!order.executed` on line 133 and the assignment of `order.executed = true` on line 157 occur very far apart; there are many external calls made between those two lines of code. Consider placing `order.executed = true` before any external calls in order to make it clear that you have avoided any possibility of a re-entrancy exploit. (In general, it is best to have internal state changes happen before external calls, if possible.)

4.2 Unnecessary Payable Modifier

The `GnosisProtocolRelayer.withdrawExpiredOrder()` function is declared as `payable`, despite it not using any provided Ether. The `payable` modifier can likely be removed from this function.

4.3 Possible Integer Truncation (2)

On line 156 of `GnosisProtocolRelayer.sol`, the expression `uint32(order.deadline/BATCH_TIME)` can yield unexpected results due to unchecked integer truncation. If the result of the expression `order.deadline/BATCH_TIME` is greater than `0xffffffff`, the results of the truncation will be zero.

Since order deadlines large enough to trigger this condition ought not to happen in practice, it may make sense to have `orderTrade()` reject unreasonably long durations.

4.4 Redundant Bounds Check

Lines 165 and 167 of `GnosisProtocolRelayer.sol` are identical; they perform the same (incorrect) bounds check. One of the two checks should be removed, and the other should be fixed (see section 3.4).

Chapter 5

Other Notes

A significant source of complexity in `GnosisProtocolRelayer` is due to the fact that it needs to consult Uniswapv2 as a price oracle across multiple transactions before it places a trade. It may be simpler to have `orderTrade()` place a limit order directly, provided that the governance inolved in calling `orderTrade()` does not require an unreasonably long voting period. The benefit of consulting an additional price oracle before placing a limit order is simply that the limit order will be placed closer to the *current* market-clearing price, rather than the price at the time the governance proposal is made. It may be simpler to speed up governance proposals rather than requiring that trades sit around long enough to obvserve stable price conditions.