SOLIDIFIED

Audit Report for Gnosis DX. March 29, 2018.

Summary

Audit Report prepared by Solidified for Gnosis covering their Dutch Exchange contract, OWL and Magnolia token contracts and upgradeable proxy contract.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the contracts. The debrief took place on March 28, 2018 and the final results are presented here.

Audited Files

The following files were covered during the audit:

- TokenMGN.sol
- PriceOracleInterface.sol
- Medianizer.sol
- DutchExchange.sol
- TokenOWL.sol
- TokenOWLProxy.sol
- ProxyMaster.sol
- OWLAirdrop.sol
- Proxy.sol

Notes

The audited versions are the ones present in the commits:

- https://github.com/gnosis/dutch-exchange/commit/8388808de1459363182a63eb27c0bcadebd4537d
- https://github.com/gnosis/owl-token/commit/32e2b76c8c3bbccaeb5de9155ac05d1ba727f50a

Intended Behavior

The purpose of the contracts is to facilitate trading of Ethereum tokens based on the principle of Dutch Auctions as described in this document:
https://drive.google.com/file/d/1OojAb6ogvQKVolkGDNVY1Pu74DbTNET6/view

Following list of known issues and weaknesses was provided by the Gnosis team:
https://drive.google.com/drive/folders/1HU3DVp-i5GJwIwAkj3f94OtzEBkFIpe2

Issues Found

Major

1. Potential abuse of auctioneer's powers

`DutchExchange` contract defines an `auctioneer` role which is an address with considerable powers over the contract. By updating the `masterCopy` variable in the proxy contract this address can completely change the behavior of the system, potentially to the users' disadvantage; because of this there's a protection in form of 30 day delay before the `masterCopy` variable update request can be fulfilled. This mechanism however can give the users false sense of security, because it doesn't effectively protect them from the worst misbehaviors of the auctioneer, described below:

1. a) Withdrawing all funds currently owned by the contract

By changing `ethUSDOracle`, the auctioneer can open the contract to a re-entrancy attack, which allows him to underflow his token balances and withdraw all tokens currently owned by the contract.

***An example of a possible attack**: `settleFee` is called in `postSellOrder`, inside `settleFee` there's `ethUSDOracle.getUSDETHPrice()` call, this function can call withdraw and lower the balance under what it was when `postSellOrder` was called, line `405` can then lower the balance under 0, thereby underflowing.*

***Note**: The fact that `getUSDETHPrice` function is defined as `view` in the interface of the oracle contract doesn't protect from calling a state changing function.*

Recommendation

Either make sure the re-entrancy attack is not possible with a re-entrancy lock, or add a delay to switching out the new `ethUSDOracle`. It should be possible to switch the contract into some sort of emergency state when `ethUSDOracle` contract fails, but there should be a time for users to react after a new `ethUSDOracle` contract is being added.

AMENDED [2018-4-11]:
Gnosis has acknowledged this is a severe issue and has taken the following countermeasures:

1. Changing the PriceOracleInterface ethUSDOracle can only be done with a 30 days delay.
2. The owner of the priceOracleInterface can trigger an emergencyMode. If this emergencyMode is triggered, the interface will report a constant price 600 USD/ETH. This will deliver an inaccurate price feed, but still a reliable price feed. For the platform an inaccurate price is not a big issue. Only auction thresholds will not be accurate and the OWL pricing might not be accurate. But it would not compromise any main functionality of the exchange.
3. Contract will double check for overflows and underflows all the time.

The issue is no longer present in commit `d7aaf39dcb72ea1d65220e41f2873da53fd450f7`.

1. b) Minting of arbitrary amount of Magnolia tokens

By creating a pair of token contracts controlled by the auctioneer, adding them to the list of approved tokens and trading them, he can effectively mint arbitrary amount of Magnolia tokens at no cost to him beyond gas expenditure. Because he can add these tokens to the approved list after he has done his trading and call `claimBuyerFunds` or `claimSellerFunds` right after, the waiting periods don't serve as a protection. By minting magnolia tokens, he can make himself the exclusive owner of the fee reducing proportion of the Magnolia tokens and recover all fees currently residing in the contract as "extra tokens" through trading.

Recommendation

Add a delay to adding tokens to the approved list to allow community to review the tokens before they are added. The removal of the tokens from the approved list can remain instant.

AMENDED [2018-4-11]:
Client's response:

Regarding Minting of arbitrary amount of Magnolia tokens:
Yes, if the auctioneer defined in the smart contract misuses his power and adds malicious tokens, he would have an advantage for collecting fees of the next auctions. But more dramatically, he would devalue all the Magnolia tokens. Since we can not prevent the more impactful devaluation of the Magnolia tokens, if we add a delay, we will not add a delay.

New issue found: Changing the TokenMGN minter to prevent claiming of auction winnings.
We found another way how the auctioneer could steal tokens: the owner of the Magnolia contract can set a minter of the Magnolia contract. This minter should be the DutchExchange.

We coded the minter address changeable since we wanted to change a minter in case we need to update the DutchExchange.
Now, the owner of the Magnolia Token contract could change the minter and thereby make any claimSellerFunds or claimBuyerFunds calls revert. Calls would revert, since they try to mint Magnolia tokens, but are not allowed. So the malicious owner of the exchange and malicious owner of the Magnolia Token could first change the minter so that no one can claim tokens and then in a second step upgrade the DutchExchange smart contract to get the power over the tokens.
The implemented fix: We hand over the ownership of the Magnolia Token from us to the DutchExchange-Proxy contract as the last step in the deployment script. Then the Magnolia minter contract could only be updated if the DutchExchange-Proxy would get a new logic which allows someone to update the minter. Hence it would be only possible after a 30-day delaying period.

Minor

2. Griefing opportunities enabled by addTokenPair

The list of known issues provided by Gnosis mentions that the starting price provided by the entity that adds a new token pair doesn't have to reflect rational market price, this problem is mitigated by enforcing minimum funding of the new auction (probably larger or equal to $10,000), but the claim that the auctions should reflect the rational market price by the end of the second auction turns out to not be necessarily true after further inspection.

The price can be set in a way such that the rational market price in the second auction falls in between the auction end time and 1 second before end time. This way, there will be no block in which the auction is still running and the price is <= rational market price. For buyers there's no purely rational way to participate in such an auction. If they decide to take the risk and enter their buy order at a higher than their internal price, they are forced to accept the price set by last accepted buy order which can also be a malicious buy order by the sellers designed to drive the final price up. Auctions with these price parameters are also prone to miner censorship attacks targeting the sellers.

One additional problem is that after the entity that created the auction with intentionally extreme initial price submits funding for the first ETH/Token pair, they can immediately create auctions with equally extreme pricing for the combinations of the attacked token with other currently traded tokens on the market with almost no additional cost. Since the contract believes that value of one unit of the token is greater than or equal to the needed threshold ($10000), the attacker only has to provide one unit of the token per trading pair.

In conclusion, auctions with extreme pricing present complicated games allowing various malicious behaviors, this might hinder auctions settling on a rational market price.

AMENDED [2018-4-11]:
Client's Response:

"The price can be set in a way such that the rational market price in the second auction falls in between the auction end time and 1 second before end time. This way, there will be no block in which the auction is still running and the price is <= rational market price. For buyers there's no purely rational way to participate in such an auction. "
It is correct that there is no meaningful way to participate as a buyer. But there is a rational way to post a BuyOrder as a seller. Seller would just post a BuyOrder of the size: their sellvolume * fair_price. If the end price is below the market price, they would make a profit from the buy. If a malicious actor pushes the price above the market price, then they would make a profit from the sell. In this way, anyone, who wants to correct a wrong, initial price could participate as a seller and buyer in the following auctions and thereby correct prices.
Because of these reasons, we will not implement any fixes for this issue.

3. Due to change of ETH/USD price, auctions that were under the starting threshold can passively surpass it

If an auction's sell-side funding surpasses the necessary auction start threshold due to a change in ETH/USD price, there is currently no way of starting the auction without submitting an additional non-zero sell order.

Recommendation

Consider decoupling scheduling the start of a new auction from posting strictly positive sell orders.

AMENDED [2018-4-11]:
Client's Response:
If there is a delay of the auction start due to this fact, it should not have any bigger impacts. We expect new sell orders for the auction from new customers to join the auction in a reasonable time.

4. Tokens based on gnosis-core-contracts/contracts/Tokens/StandardToken.sol (i.e. the audited tokens) are vulnerable to approve attack

It's recommended that `StandardToken` implements one of the ERC20 compatible mitigations of the known EIP20 API Approve / TransferFrom multiple withdrawal attack: https://github.com/ethereum/EIPs/issues/738.

AMENDED [2018-4-11]:
Client's Response:
We do not see this as an issue, since in 99% of the use cases, one gives the approvals only to trusted contracts.

Note

5. Each approved token contract is a potential Magnolia minting attack vector

Approved token contract, if compromised or misused by its owner, can be used for minting Magnolia tokens, in the extreme case of two token contracts compromised at once, an arbitrary amount of magnolia can be minted in two blocks at almost no cost. With the platform growing and allowing trading of large amount of tokens, the probability of similar attack increases, mitigations of the attack should be considered alongside possible recovery strategies.

AMENDED [2018-4-11]:
Client's Response:
Yes, Magnolia holders need to trust the auctioneer of the exchange not to approve malicious tokens. But the risk is only a devaluation of Magnolia tokens, tokens on the exchange should be safe - besides small fee amounts.

6. Sellers are disproportionately affected by network congestion

It's rational for buyers to include gas costs in the calculation of the effective price they buy the tokens at: to ensure it is <= their internal price. If there's a rise in average gas price, for example in case of network congestion, all buy order prices will be shifted lower at the expense of the sellers.

AMENDED [2018-4-11]:
Client's Response:
Right, but no changes.

7. Consider adding pre-approval of max fee and max price

Since the time and order in which the transactions get included in a block can be unpredictable, it could be useful to allow users to pre-approve values of attributes that depend on time and order. Such as maximum auction price and maximum fee percentage. These assurances can be achieved by submitting transactions form a third-party contract, but they could also be provided as convenience functions inside the core contract.

AMENDED [2018-4-11]:
Client's Response:
Max price is already implemented, as one specifies the highest price in an auction by waiting until the price is reached and then only send the transaction.

Solidified Response
Implementing max fee could still be useful, in case of max price our reasoning is that the user could want to send the buy order ahead of the target block and the max price parameter is an assurance that the buy will not be included before the target block. Neither functionality is crucial though.

8. Redundant arguments

8.a Since `claimAndWithdraw` function throws if `user != msg.sender` the user argument seems redundant.

AMENDED [2018-4-11]:
Client's Response:
We do not consider this as an issue, since one might claim funds for someone else and do a withdraw for your own account.

8.b `settleFee` is an internal function and it's always called with `user = msg.sender`; therefore, the argument could be omitted without loss of function and replaced by `msg.sender` in the body of the function.

AMENDED [2018-4-11]:
The issue is no longer present in commit `d7aaf39dcb72ea1d65220e41f2873da53fd450f7`.

9. Consider indexing event arguments

If searchable logs are desirable, the events should have indexed arguments (with caveat that indexed arguments are not directly recoverable).
https://solidity.readthedocs.io/en/develop/contracts.html#events

For example order events, such as this one:

```
event NewSellOrder(
    address sellToken,
    address buyToken,
    address user,
    uint auctionIndex,
    uint amount
);
```

could potentially be indexed by `sellToken`, `buyToken`, and/or `user`.

AMENDED [2018-4-11]:
Client's Response:
Good advice. Thanks

10. Consider validating input arrays are same length in function claimTokensFromSeveralAuctions

Though not a security issue, for consistency and usability it might be a good idea to validate that `auctionSellTokens`, `auctionBuyTokens`, and `auctionIndices` are all the same length.

```
function claimTokensFromSeveralAuctions(
        address[] auctionSellTokens,
        address[] auctionBuyTokens,
        uint[] auctionIndices,
        address user
    )
        public
    {
        for (uint i = 0; i < auctionSellTokens.length; i++)
            claimSellerFunds(auctionSellTokens[i], auctionBuyTokens[i],
user, auctionIndices[i]);
```

```
    }
```

AMENDED [2018-4-11]:
Check for arrays length was added by the Gnosis team in commit `d7aaf39dcb72ea1d65220e41f2873da53fd450f7`.

11. Last Buy-Order does not pay fees

The buy order that closes an auction is not charged fees. This does not currently appear in the spec, so documentation for this should be added.

AMENDED [2018-4-11]:
Client's Response
Gnosis team will update the documentation spec to reflect this.

12. Misc.

- DutchExchange.sol:437 Variable `auctionStart` set on line 431 can be used instead of `getAuctionStart(sellToken, buyToken)`

AMENDED [2018-4-11]:
The above issue (bullet #1) is no longer present in commit `d7aaf39dcb72ea1d65220e41f2873da53fd450f7`.

- DutchExchange.sol:775 Expression in condition is always true because of 446
- DutchExchange.sol:460 Can't be lower than zero because of how `getCurrentAuctionPrice` works

13. It's good practice to set visibility of purely external functions to "external" instead of "public"

In case of functions that are never called inside of the contract, setting their visibility to `external` instead of `public` can provide gas savings in some situations. Though that's not necessarily the case in the audited contracts, it's still good practice to do so.

AMENDED [2018-4-11]:
This issue is no longer present in commit `d7aaf39dcb72ea1d65220e41f2873da53fd450f7`.

14. Funds locked in the contract due to rounding

It's possible that the contract will accumulate irretrievable token balances due to division rounding present at various places. These amounts are expected to be negligible.

15. Miner of block that potentially includes an auction closing buy order has limited front-running ability

Miner's have complete control over which transactions they accept and in what order they're executed. This means that when they mine a block that could include a closing buy (from themselves or others), they have the following privileges:

- If the miner made a buy during a previous block in the same auction, they can censor closing buy orders at virtually 0 cost to themselves and gain a small amount of tokens by doing so. While the price is increasingly resistant to manipulation by censorship over a number of blocks, it's still rational in the marginal case for individual miners to censor buy orders.
- Given enough liquidity in the buy-side token, the miner could effectively enforce right of first refusal for closing buy orders. They can selectively accept and arrange buy orders to ensure their own order closes the auction, potentially unfairly bumping others out of the buy window.

AMENDED [2018-4-11]:
Client's Response
Mentioned points are right. Other exchanges have a bigger attack surface for front-running.

16. Overflow not entirely handled

There is no particular protection against overflows in both `totalTokens` and `lockedTokenBalances` in the `mintTokens()` on the MNG contract. Though this is only probable when Magnolia minting is compromised.
Additionally balances are implicitly assumed to be < 10 ** 30: the deposit function only checks that the deposited amount doesn't overflow balance, and the order posting functions accepts any amount up to your balance. Since potential overflows aren't checked further (only that the results of operations involving balance < 10 ** 30, which would be true in case of overflow), various operations in the order can be caused to overflow. Again, this is only probable in case of malicious or compromised tokens, so potential damage appears to be limited.

AMENDED [2018-4-11]:
All mathematical operations are handled by SafeMath in commit `d7aaf39dcb72ea1d65220e41f2873da53fd450f7`.

17. Consider implications of magnolia pooling

Magnolia tokens can be pooled together in a smart contract and locked collectively for trustless crowdfunded fee reduction. In one possible scenario, buyers place their orders through the smart contract, and any magnolia generated stays in the pool. This is rational for buyers as long as (sale value of magnolia tokens) < (personal fee rate - pool fee rate) * (total value of buy) - cost for extra gas of pool buying. For markets that don't generate magnolia, the fee reduction only has to beat the cost for extra gas. More complicated arrangements with stronger incentives, such as magnolia contributors being rewarded with fees, can be imagined.

AMENDED [2018-4-11]:
Client's Response
There is no to little incentive to share you own Magnolia with others, if the result is that one earns less fees. We do not consider it as an issue right now. In case this becomes an issue in the future, we will update the DX-smartcontract with the proxy solution.

Our Response
We agree that the likelihood of magnolia pooling on an impactful scale is low, and immediate action is not necessary. However, consider the following scenario: a user that does large volume of trades in magnolia producing markets, but doesn't trade other coins on DutchX can collect their own rent on those other markets by sharing access to their locked tokens. They were going to receive 0 fees from those markets because they do not trade in them (they would only allow buys in markets they don't plan on participating in), but now they receive a reduced fee from any market participant who stands to pay a lower fee. So the actor who can provide the largest fee reduction stands to gain all fees in a given market (minus the discount they provide).

Closing Summary

All major reported issues have been fixed or consciously addressed and chosen to be left as is, taking into consideration the engineering trade-offs.

Additional helper view functions and minor edits were made between the two rounds of audit, and were verified as not imposing any security risk.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the Gnosis brand or its products. This audit does not provide a security or correctness guarantee of the audited smart contracts. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.