SOLIDIFIED

Audit Report for Polymath. March 22, 2019.

Summary

This audit has been limited to the VolumeRestrictionTransferManager module, release 2.1.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the below token swap. The debrief took place on March 22, 2019 and the final results are presented here.

Audited Files

- VolumeRestrictionTM.sol
- VolumeRestrictionTMFactory.sol
- VolumeRestrictionTMStorage.sol

The files were audited on commit `a8b71e575526284c08803e156ab0c3feca198989` and solidity compiler version `0.4.24`

Intended Behavior

The purpose of these contracts is to implement the Volume Restriction Transfer Manager module.

Issues Found

Critical

No critical issues were found.

Major

1. Possible accounting corruption between `individualRestriction` and `defaultRestriction`

If:
1) `individualRestriction` and `defaultRestriction` have the same period offset, meaning `individualRestriction.startTime % 1 day == defaultRestriction.startTime % 1 day`

and

2) `individualRestriction` ends inside `defaultRestriction`, meaning `individualRestriction.endTime >= defaultRestriction.startTime && individualRestriction.endTime < defaultRestriction.endTime`

It's possible for `individualRestriction` and `defaultRestriction` to touch the same buckets after `individualRestriction` ends, which will create problems in `_bucketCheck()` function on line 956 where `sumOfLastPeriod` of `defaultUserToBucket` which hasn't been incremented by transactions that happened inside `individualRestriction` will be decreased by these transactions when subtracting past no longer relevant buckets. This can lead to underflow blocking the execution of the contract, or effectively increase `defaultRestriction` limit for the user.

Recommendation
Consider separating bucket accounting of default and individual restrictions.

Minor

2. Possible accounting corruption between consecutive restrictions

If inside one block
1) restricted transaction happens with `fromTimestamp == now`

2) restriction is removed through `removeDefaultRestriction()` or `removeIndividualRestriction()`
3) new restriction of the same type is added with `startTime == now`

The new restriction will end up sharing its first bucket with the last bucket of the previous restriction, leading to corrupted accounting in `_bucketCheck()` (see issue #1).

Recommendation
Consider either

a) preventing deletion of restrictions until they are expired,
b) or enforcing `_startTime > now` for new restrictions to prevent bucket overlap and removing:

```
require(
    restriction[_holder].endTime < now,
    "Not Allowed"
);
```

from addRestriction functions to make deletion requirements consistent across the contract.

Notes

3. Use external functions

It's recommended that interfacing functions are market as external for both clarity and consistency. Although some functions are already marked as such, there are others that can be modified, like `addIndividualRestrictionMulti`, `addIndividualDailyRestrictionMulti` and `changeExemptWalletList`.

4. Reasons for revert are not descriptive

Most of the error messages in the contract are simply `"not allowed"`, which does not provide much information about the error itself. It's recommended to use descriptive error messages.

5. Avoid repetitive code

Some functions can be abstracted to avoid code repetition. For example, `defaultRestrictionCheck` and `IndividualRestrictionCheck` are almost identical and its difference could be passed as a parameter. This will make bytecode smaller as well as avoid future mistakes when modifying the functionality.

6. ITransferManager contains implementation

It's widely accepted in the community that contracts prefixed with "I" are interfaces that do not contain implementation.

7. Prefix parameters with "_" if function is internal

Some of the internal functions do not comply with the determined code style of prefixing parameters name with `"_"`, like `_dailyTxCheck` and `_setValues`.

8. VolumeRestrictionTM: restrictionType is passed as uint256 in several occasions

The type definition `enum RestrictionType { Fixed, Percentage }` could be used instead for clarity.

9. VolumeRestrictionTM: Incorrect commenting of addIndividualDailyRestriction

According to the comments, this function is used to add the new individual daily restriction for all token holder. However, the implemented functionality applies restrictions to individual holders.ead for clarity.

Conclusion

One major issue was found, along with some minor issues that can affect the desired behavior and it's recommended that they're addressed before proceeding to deployment.

Disclaimer