SOLIDIFIED

Audit Report for Polymath Token Distribution. January 9, 2018.

Summary

Audit Report prepared by Solidified for Polymath covering the Token Distribution contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the below token distribution contracts. The debrief and cross-verification took place on January 9, 2018 and the final results are presented here.

Audited Files

The following files were covered during the audit:

* **[PolyDistribution.sol]** (./contracts/PolyDistribution.sol)
* **[PolyToken.sol]** (./contracts/PolyToken.sol)
* **[Ownable.sol]** (./contracts/Ownable.sol)
* **[SafeMath.sol]** (./contracts/SafeMath.sol)

Intended Behavior

The purpose of the contracts is to distribute and manage polymath tokens between multiple owners and contracts according to this schedule:

- Presale supply - 240M (no vesting no cliff)
- Founder - 150M (4 year vest, 1 year cliff)
- Airdrop - 10M (no vesting, no cliff)
- Advisor - 25M (no vesting, 7 month cliff)
- Bonus - 80M (4 year vest, 1 year cliff) assigned to presale investors proportional to how much they purchased
- Reserve - 495M (4 year vest, 6 month cliff)

Each 'vest' drips out a linear amount over time from the `startTime` until the cliff.

Solidified Stamp

Polymath must address issues (1) and (2) and potentially (3) - depends on intended behaviour - in the final version of the contract in order to qualify for a Solidified stamp of approval.

SOLIDIFIED

Audit Report for Polymath Token Distribution. January 9, 2018.

1. Owner of PolyDistribution can bypass allocation/vesting and drain all POLY held by PolyDistribution

---

PolyDistribution.sol / line 113

```
require(_token != address(this));
```

Supposed intent behind this check in the `refundTokens` function is to prevent owner from using the function to transfer POLY tokens, but instead of the POLY token address, the address of the distribution contract is used, so the check has no practical effect. The correct version of the code is:

```
require(_token != address(POLY));
```

AMENDED [2018-1-22]:

This issue has been fixed by the Polymath team and is no longer present in commit aa042b7.

2. Tokens allocated from `AVAILABLE_PRESALE_SUPPLY` and `AVAILABLE_AIRDROP_SUPPLY` can be transferred before "vesting clock" has started

---

PolyDistribution.sol / lines 21, 23

```
uint256 public AVAILABLE_PRESALE_SUPPLY  = 240000000 * decimals;

uint256 public AVAILABLE_AIRDROP_SUPPLY  = 10000000 * decimals;
```

Since there is no check that now >= startTime in `transferTokens`, tokens allocated from `AVAILABLE_PRESALE_SUPPLY` and `AVAILABLE_AIRDROP_SUPPLY` can

be transferred the instant they're allocated; not "Released at Token Distribution (TD)" as it states in the comment.

`require(now >= startTime);` should be added to the top of `transferTokens`.

AMENDED [2018-1-22]:

This issue has been fixed by the Polymath team and is no longer present in commit aa042b7.

3. Allocations can occur after `startTime`

PolyDistribution.sol / lines 61-87

```
function setAllocation (address _recipient, uint256 _totalAllocated, uint8 _supply)
onlyOwner public {

    require(allocations[_recipient].totalAllocated == 0 && _totalAllocated > 0);
```

The intended behavior of `setAllocation` after `startTime` has passed is unclear. This may or may not be intentional. If allowing allocations after startTime is valid, then a comment explicitly stating so should be added. Otherwise, `require(startTime > now);` should be added to the top of `setAllocation`.

AMENDED [2018-1-22]:

This issue has been addressed by the Polymath team: allocations after `startTime` are intended behavior.

4. State variable `grandTotalAllocated` can be converted into function with view modifier to save gas

`grandTotalAllocated` can be easily calculated from `AVAILABLE_TOTAL_SUPPLY`, so keeping it as state and updating it on every allocation is wasteful.

Remove `grandTotalAllocated` the state variable and replace with a function to the effect of

```
function grandTotalAllocated() public view returns (uint256) {
    return 1000000000 * 10 ** 18 - AVAILABLE_TOTAL_SUPPLY;
}
```

AMENDED [2018-1-22]:

This issue has been fixed by the Polymath team and is no longer present in commit aa042b7.

5. Consider renaming `decimals` in PolyDistribution

PolyDistribution.sol / line 18

```
uint256 private constant decimals = 10**uint256(18);
```

Since "decimals" has a specific definition in the ERC20 standard (the amount of decimal places), the non-standard usage seen below is likely to cause confusion for those reviewing the contract.

While not a security concern, for the sake of clarity, `decimalFactor` is a more appropriate name for this constant.

AMENDED [2018-1-22]:

This issue has been fixed by the Polymath team and is no longer present in commit aa042b7.

6. Log arguments are not indexed

PolyDistribution.sol / lines 41, 42

```
event LogNewAllocation(address _recipient, uint8 _fromSupply, uint256
_totalAllocated, uint256 _grandTotalAllocated);

event LogPolyClaimed(address _recipient, uint8 _fromSupply, uint256 _amountClaimed,
uint256 _totalAllocated, uint256 _grandTotalClaimed);
```

Depending on the way the authors want to work with the logs, _fromSupply and maybe even _recipient arguments could be indexed to make the logs seachable, for more info see: **https://solidity.readthedocs.io/en/develop/contracts.html#events**. Keep in mind that values of indexed arguments are not retrievable, their sole function is to provide ability to filter the log entries.

AMENDED [2018-1-22]:

This issue has been fixed by the Polymath team and is no longer present in commit aa042b7.

7. Include licenses of 3rd party code

Various OpenZeppelin contracts have been included in the repo by copying rather than by package manager. This is not recommended by OpenZeppelin, though not necessarily incorrect as npm, etc. can be considered an attack vector. That said these contracts are under the MIT license, which requires its license/copyright notice to be included along with the code.

AMENDED [2018-1-22]:

This issue has been fixed by the Polymath team and is no longer present in commit aa042b7.

8. Preference for using the AllocationType enum

Two data structures are used to represent the type of the allocation throughout the contract: `uint8` and an `enum` and although it does not affect the contract functionality, it would be more consistent to use only one, as it would improve the code readability. Recommend using enum, because it avoids the conversion and preserves the API (since when calling this function from the outside, the caller would need to pass the integer that converts to allocation type).

AMENDED [2018-1-22]:

This issue has been fixed by the Polymath team and is no longer present in commit aa042b7.

9. Avoid storing non-fundamental data

Contract storage is expensive and therefore only recommended for data important to the contract logic. Both `grandTotalAllocation` and `grandTotalClaimed` serve no

fundamental purpose and could be calculated off chain, making use of events, and saving ETH in the long term.

Closing Summary

PolyToken.sol has been verified as fully ERC20 compliant, and has taken recommended measures to mitigate the known [EIP20 API Approve / TransferFrom multiple withdrawal attack] (**https://github.com/ethereum/EIPs/issues/738**).

Beyond the issues mentioned, the contracts were also checked for overflow/underflow issues, DoS, and re-entrancy vulnerabilities. None were discovered.

OpenZeppelin contracts such as Ownable/SafeMath have been widely audited and secured, as such, they were not prioritized for auditing.

AMENDED [2018-1-22]:

The Polymath team has addressed all major issues reported in the original audit, and no further issues were found as of commit aa042b7.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the Polymath platform. 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.