Polymath Audit

FEBRUARY 13, 2018 | IN SECURITY AUDITS | BY OPENZEPPELIN SECURITY

The Polymath team asked us to review and audit their POLY Token contracts. We looked at the code and now publish our results.

The audited code is located in the polymath-token-distribution repository. The version used for this report is commit `672fabe081e8f90ea025252d92c2eb247d60010e`.

Here is our assessment and recommendations, in order of importance.

Update: *The Polymath team has followed most of our recommendations and updated the contracts. The new version is at commit* `0b47ae467f95a02c6b71421e5816b5d50b698158`.

Critical Severity

No issues of critical severity.

High Severity

No issues of high severity.

Medium Severity

Possible overflow in loop index variable

The `airdropTokens` function of the `PolyDistribution` contract takes an array of addresses as a parameter in order to "airdrop" tokens to each of them. To do so, a `for` loop is used, with an index variable of type `uint8`. This will result in an overflow for arrays which have more than 255 addresses. The loop will indefinitely iterate over the first 255 addresses, eventually failing with an out of gas error, wasting gas.
Consider using a `uint256` variable for the index.
***Update**: Fixed in commit `0b47ae4`.*

Low Severity

Incomplete ERC20 Interface

The `IERC20` contract defines the basic interface of a standard token to be used by the `PolyToken` contract. However, this contract doesn't follow the ERC20 standard which requires for the `totalSupply` function to be defined in its public interface.
We recommend dropping this contract in favor of the `ERC20` contract from the OpenZeppelin library. If not, consider adding the missing function to the contract to comply with the standard and making this contract an interface as its name and usage suggests.
***Update**: Contract is now an interface `0b47ae4`.*

Install OpenZeppelin via NPM

The `SafeMath` and `Ownable` contacts were copied from the OpenZeppelin repository, and `PolyToken` is a copy of the `StandardToken` contract. Consider making the `PolyToken` contract inherit from `StandardToken` to minimize its logic, and following the recommended way to use OpenZeppelin contracts, which is via the zeppelin-solidity NPM package, allowing for any bugfixes to be easily integrated into the codebase.

No Transfer event for minted tokens

It is recommended, in the ERC20 spec, to emit a `Transfer` event with the source (`_from`) set to `0x0` when minting new tokens. This enhances user experience by allowing applications such as Etherscan to learn of the new token holders. In this case this is only relevant for the constructor, where the initial balance is assigned to the distribution contract. Nonetheless, consider emitting the corresponding event: `Transfer(0x0, msg.sender, _initialAmount)`.
***Update**: Fixed in commit `0b47ae4`.*

Token distribution address can be null

In the `PolyToken` constructor, the total supply of the token is granted to the `_polyDistributionContractAddress`, which as its name suggests, it is expected to be the `PolyDistribution` contract's address.
However, this parameter is never inspected nor validated, allowing it to be the `0x0` address, which would make the contract unusable. This goes against the principle of *fail early and loudly*.
Consider prohibiting the null address as a parameter of the `PolyToken` constructor.
***Update**: Fixed in commit `0b47ae4`.*

Notes & Additional Information

- In the `PolyToken` and `PolyDistribution` contracts there are several numbers with too many digits, making them hard to read and error-prone. We recommend replacing them with their scientific notation equivalents. For example, `10e9` for `PolyTokens`'s `totalSupply`.
- There is a transfer of tokens in the function `transferTokens` whose return value is unchecked. Even though the token as of now never returns `false`, it is good practice to not omit the check, as was correctly done in the rest of the contract. *(**Update:** Fixed in `0b47ae4`.)*

Conclusion

No critical or high severity issues were found. Some changes were proposed to follow best practices and reduce potential attack surface.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the POLY Token contracts. We have not reviewed the related Polymath project. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.

Security Audits

- If you are interested in smart contract security, you can continue the discussion in our forum, or even better, join the team 🚀
- If you are building a project of your own and would like to request a security audit, please do so here.

RELATED POSTS

SECURITY AUDITS

Solidity Compiler Audit

The Augur team and the Ethereum Foundation (through a joint grant) asked us to review and audit the...

READ MORE

by OpenZeppelin Security

SECURITY AUDITS

External audit of OpenZeppelin

As you may know from reading our blog, we do lots of security audits for blockchain-based projects....

READ MORE

by OpenZeppelin Security