CERTIK

Security Assessment

Polylastic - Airdrop and Token Swap

May 22nd, 2021

Table of Contents

Summary

This report has been prepared for Polylastic - Airdrop and Token Swap smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Majority of the findings are of informational nature with four minor and one medium findings. Minor findings comprise the lack of validation of constructor parameters and unchecked values of tokens transfers while the medium finding relates to the possibility of reentrancy attack.

Overview

Project Summary

Project Name	Polylastic - Airdrop and Token Swap
Description	The audited contracts comprise Airdrop and TokenSwap contracts. Airdrop contract airdrops tokens to all the users using a signature and TokenSwap contracts allows users to send their old POLX tokens and get the new one in 1:1 ratio.
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/polylastic/smart-contracts
Commits	1. 616d4cd3f6d3d7807e50e61c595001f24dfd7828 2. 23cba73f8bcdb704b74b1dd20bb36e0f06909cf3

Audit Summary

Delivery Date	May 22, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

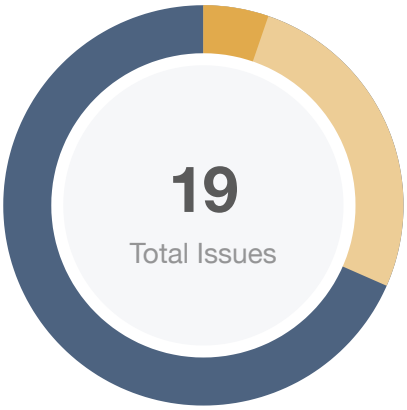
Vulnerability Summary

Total Issues	19
Critical	0
Major	0
Medium	1
Minor	5
Informational	13
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
AIR	Airdrop.sol	ddb779ecabe78745e8e9b717d8c0d6598074eb667347f499e6db6f85df1407e1
TSP	TokenSwapPortal.sol	0e36fc9851d1850eff69a81e097d62b9635d04ab639846d1e2b3e10ce25e75a6

Findings



Critical	0 (0.00%)
Major	0 (0.00%)
Medium	1 (5.26%)
Minor	5 (26.32%)
Informational	13 (68.42%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
AIR-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
AIR-02	Library `ECDSA` is attached with any data type	Language Specific	Informational	Resolved
AIR-03	Mutability Specifiers Missing	Gas Optimization	Informational	Declined
AIR-04	Redundant Statements	Inconsistency	Informational	Resolved
AIR-05	Lack of validation for constructor parameter	Logical Issue	Minor	Resolved
AIR-06	Comparison with boolean literal	Gas Optimization	Informational	Resolved
AIR-07	Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Call	Volatile Code	Minor	Resolved
AIR-08	Possibility of reentrancy attack	Volatile Code	Medium	Resolved
AIR-09	SPDX license identifier not provided	Language Specific	Informational	Declined
TSP-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
TSP-02	Library `SafeMath` is attached with any data type	Language Specific	Informational	Resolved

ID	Title	Category	Severity	Status
TSP-03	Mutability Specifiers Missing	Gas Optimization	Informational	Declined
TSP-04	Lack of validation for constructor parameter	Logical Issue	Minor	Resolved
TSP-05	Unchecked Value of ERC-20 `transfer()` Call	Volatile Code	Minor	Resolved
TSP-06	Inexistent Error Messages	Coding Style	Informational	Resolved
TSP-07	Token amount is not validated against zero	Volatile Code	Informational	Resolved
TSP-08	Token amount is not validated against zero	Volatile Code	Informational	Declined
TSP-09	SPDX license identifier not provided	Language Specific	Informational	Declined
TSP-10	Unchecked Value of ERC-20 `transfer()` Call	Volatile Code	Minor	Resolved

AIR-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	Airdrop.sol: 1	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for `^0.6.12`, the version can be locked at `0.6.12`.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3`.

AIR-02 | Library `ECDSA` is attached with any data type

Category	Severity	Location	Status
Language Specific	Informational	Airdrop.sol: 8	Resolved

Description

The aforementioned line attaches `ECDSA` library with any data type yet the library is declared to work with `bytes32` data type and is used with the `bytes32` type variables in the codebase.

Recommendation

We recommend to explicitly attach the library `ECDSA` only with the data type `bytes32` to increase the legibility of the codebase.

```
using ECDSA for bytes32;
```

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3`.

AIR-03 | Mutability Specifiers Missing

Category	Severity	Location	Status
Gas Optimization	Informational	Airdrop.sol: 10, 12, 13	Declined

Description

The linked variables are assigned to only once, either during their contract-level declaration or during the `constructor`'s execution.

Recommendation

For the former, we advise that the `constant` keyword is introduced in the variable declaration to greatly optimize the gas cost involved in utilizing the variable. For the latter, we advise that the `immutable` mutability specifier is set at the variable's contract-level declaration to greatly optimize the gas cost of utilizing the variables. Please note that the `immutable` keyword only works in Solidity versions `v0.6.5` and up.

Alleviation

No alleviations.

AIR-04 | Redundant Statements

Category	Severity	Location	Status
Inconsistency	Informational	Airdrop.sol: 14	Resolved

Description

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation

We advise that they are removed to better prepare the code for production environments.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3` by utilizing the variable in code.

AIR-05 | Lack of validation for constructor parameter

Category	Severity	Location	Status
Logical Issue	Minor	Airdrop.sol: 20	Resolved

Description

The parameters of constructor on the aforementioned line are used to initialize the state of variables of contract and yet not validated against zero value. As the initialized state variables of the contract cannot be changed later, it can result in unwanted state of the contract.

Recommendation

We recommend to validate the address type constructor parameters on the aforementioned lines against zero address value for the address type variables and against integer zero for unsigned integer type variable.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3`.

AIR-06 | Comparison with boolean literal

Category	Severity	Location	Status
Gas Optimization	Informational	Airdrop.sol: 29, 30	Resolved

Description

The aforementioned lines perform comparison with boolean literal which can substituted with either with the expression or the negation of the expression to save gas cost.

Recommendation

We advise to substitute the comparison with boolean literal on `L29` with the expression itself and on `L30` with the negation of the expression.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3`.

AIR-07 | Unchecked Value of ERC-20 `transfer()`/`transferFrom()` Call

Category	Severity	Location	Status
Volatile Code	Minor	Airdrop.sol: 32	Resolved

Description

The linked `transfer()`/`transferFrom()` invocations do not check the return value of the function call which should yield a `true` result in case of a proper ERC-20 implementation.

Recommendation

As many tokens do not follow the ERC-20 standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` and `transferFrom()` functions of ERC-20 tokens. The OZ implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3` by explicitly checking the return value of `transfer` function.

AIR-08 | Possibility of reentrancy attack

Category	Severity	Location	Status
Volatile Code	Medium	Airdrop.sol: 32, 35	Resolved

Description

The aforementioned line performs transfer of airdrop tokens to the user. An `ERC20` and `ERC777` implementations, such as tokens like `imBTC` can inform the recipient of token transfer with callback call thus leading to re-entrancy possibility. If the function is re-entered then the `msg.sender` is able to drain all the airdrop tokens balance of the contract as the `claimed` status of the user is set to `true` after the `transfer` call.

Recommendation

We advise to update the `claimed` status of the user to `true` before the `transfer` call, so any re-entry attempt is reverted by the function.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3`.

AIR-09 | SPDX license identifier not provided

Category	Severity	Location	Status
Language Specific	Informational	Airdrop.sol: 1	Declined

Description

The source file does not specify SPDX license identifier.

Recommendation

Consider adding the SPDX license identifier before deployment

Alleviation

No alleviation.

TSP-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	TokenSwapPortal.sol: 1	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for `^0.6.12`, the version can be locked at `0.6.12`.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3`.

TSP-02 | Library `SafeMath` is attached with any data type

Category	Severity	Location	Status
Language Specific	Informational	TokenSwapPortal.sol: 9	Resolved

Description

The aforementioned line attaches `SafeMath` library with any data type yet the library is declared to work with `uint256` data type and is used with the `uint256` type variables in the codebase.

Recommendation

We recommend to explicitly attach the library `SafeMath` only with the data type `uint256` to increase the legibility of the codebase.

```
using SafeMath for uint256;
```

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3`.

TSP-03 | Mutability Specifiers Missing

Category	Severity	Location	Status
Gas Optimization	Informational	TokenSwapPortal.sol: 11, 13, 15	Declined

Description

The linked variables are assigned to only once, either during their contract-level declaration or during the `constructor`'s execution.

Recommendation

For the former, we advise that the `constant` keyword is introduced in the variable declaration to greatly optimize the gas cost involved in utilizing the variable. For the latter, we advise that the `immutable` mutability specifier is set at the variable's contract-level declaration to greatly optimize the gas cost of utilizing the variables. Please note that the `immutable` keyword only works in Solidity versions `v0.6.5` and up.

Alleviation

No alleviations.

TSP-04 | Lack of validation for constructor parameter

Category	Severity	Location	Status
Logical Issue	Minor	TokenSwapPortal.sol: 19	Resolved

Description

The address type parameters of constructor on the aforementioned line are used to initialize the state of variables of contract and yet not validated against zero address value. As the initialized state variables of the contract cannot be changed later, it can result in unwanted state of the contract.

Recommendation

We recommend to validate the address type constructor parameters on the aforementioned lines against zero address value.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3`.

TSP-05 | Unchecked Value of ERC-20 `transfer()` Call

Category	Severity	Location	Status
Volatile Code	Minor	TokenSwapPortal.sol: 33, 44	Resolved

Description

The linked `transfer()` invocations do not check the return value of the function call which should yield a `true` result in case of a proper ERC-20 implementation.

Recommendation

As many tokens do not follow the ERC-20 standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` function of ERC-20 tokens. The OZ implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3` by explicitly checking the return value of `transfer` operations.

TSP-06 | Inexistent Error Messages

Category	Severity	Location	Status
Coding Style	Informational	TokenSwapPortal.sol: 42, 43	Resolved

Description

The linked `require` checks do not contain any error message specified.

Recommendation

We advise the error messages of these checks to be set to properly illustrate what the conditionals within evaluate.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3`.

TSP-07 | Token amount is not validated against zero

Category	Severity	Location	Status
Volatile Code	Informational	TokenSwapPortal.sol: 31	Resolved

Description

The token amount on the aforementioned line is used in ERC-20 transfers and yet it is not validated against zero.

Recommendation

Consider adding a requirement that the sender's balance amount should be non-zero after line `L31`.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3`.

TSP-08 | Token amount is not validated against zero

Category	Severity	Location	Status
Volatile Code	Informational	TokenSwapPortal.sol: 39	Declined

Description

The token amount on the aforementioned line is used in ERC-20 transfers and yet it is not validated against zero.

Recommendation

Consider adding a requirement that the value of the supplied `amount` parameter should be non-zero.

Alleviation

No alleviations.

TSP-09 | SPDX license identifier not provided

Category	Severity	Location	Status
Language Specific	Informational	TokenSwapPortal.sol: 1	Declined

Description

The source file does not specify SPDX license identifier.

Recommendation

Consider adding the SPDX license identifier before deployment.

Alleviation

No alleviations.

TSP-10 | Unchecked Value of ERC-20 `transfer()` Call

Category	Severity	Location	Status
Volatile Code	Minor	TokenSwapPortal.sol: 32	Resolved

Description

The linked `transfer()` invocations do not check the return value of the function call which should yield a `true` result in case of a proper ERC-20 implementation.

Recommendation

As many tokens do not follow the ERC-20 standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` function of ERC-20 tokens. The OZ implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

Alleviation

Alleviations are applied as of commit hash `23cba73f8bcdb704b74b1dd20bb36e0f06909cf3` by explicitly checking return value of `transferFrom` function.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.