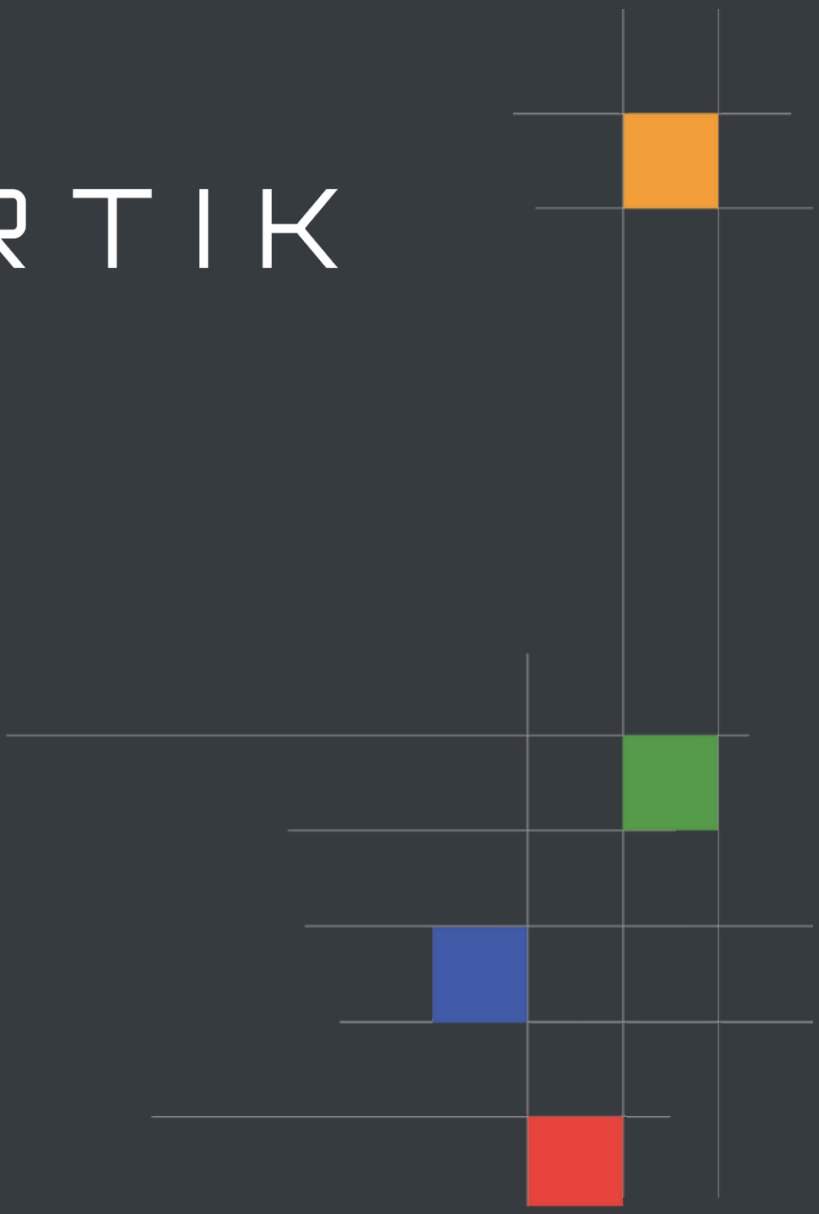
Polylastic

Security Assessment

May 17th, 2021



Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are 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 review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Polylastic
Description	A typical ERC20 implementation
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. bf2fd52633058254cd59da37d1b4f1f0114df0ae 2. 04679d19bb869505ac011b2bfcd7a20eeb742524

Audit Summary

Delivery Date	May 17th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	April 29th, 2021 - April 30th, 2021

Vulnerability Summary

Total Issues	13
Total Critical	0
Total Major	0
Total Medium	1
Total Minor	3
Total Informational	9

Executive Summary

Polylastic Token is standard ERC20 token. All contracts are taken from OpenZeppelin version 3.4.0 with added functionality to mint total supply of tokens to the owner of a contract. PolylasticToken.sol contract differs in one area from OpenZeppelin ERC20.sol contract. The client removed `_beforeTokenTransfer` internal function which in OZ implementation was virtual. PolylasticTokenV2.sol contract implements fee for transfering tokens and also add whitelist functionality. As we suggested in the finding, we recommend importing Ownable.sol contract instead of implementing only some functions. It gives more control over the owner and if there is a need, also to renounce ownership.

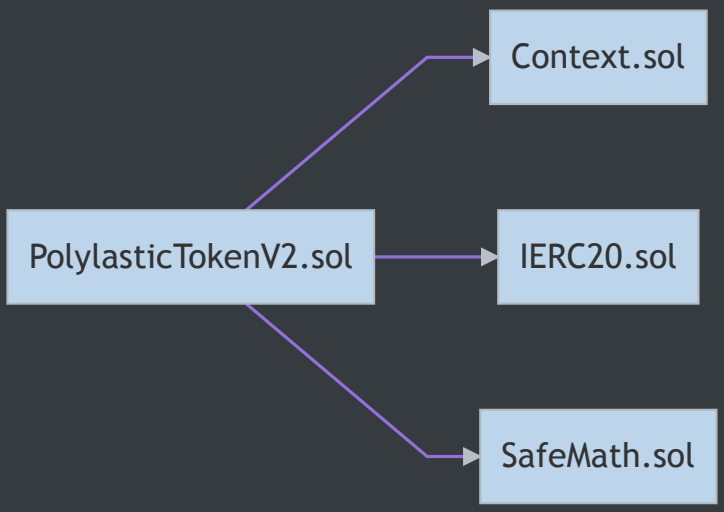
System Analysis

Contract owner gets total supply of the tokens. We weren't notify what is the purpose of the token or where it's gonna be used but this rises point of centralization.
With V2 of the Token, same concern is raised along side with only admin functions.

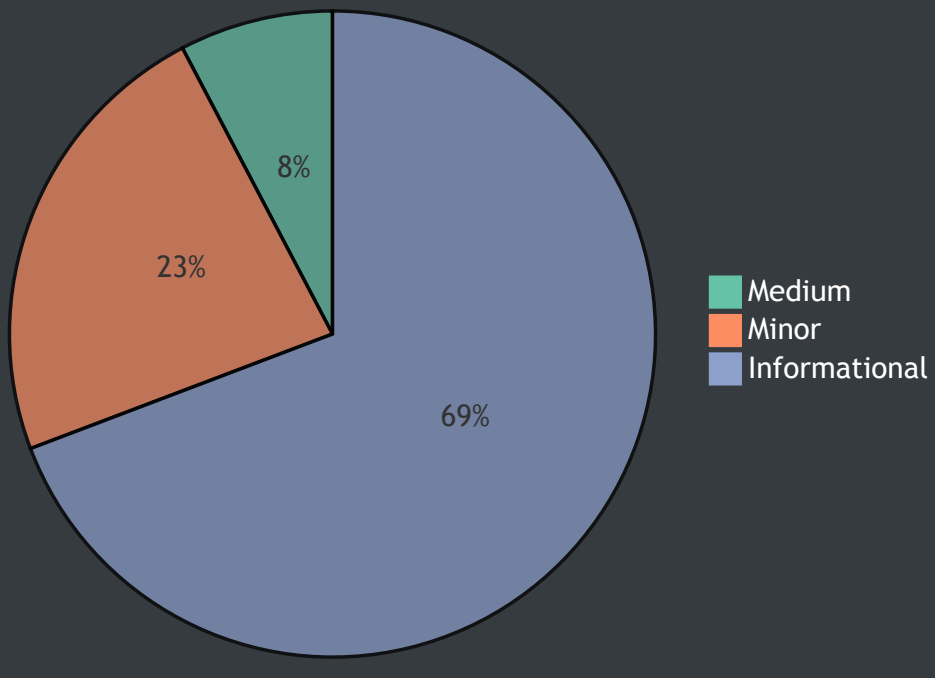
Files In Scope

ID	Contract	Location
IER	IERC20.sol	contracts/IERC20.sol
PTN	PolylasticToken.sol	contracts/PolylasticToken.sol
PTV	PolylasticTokenV2.sol	contracts/PolylasticTokenV2.sol
SMH	SafeMath.sol	contracts/math/SafeMath.sol
CON	Context.sol	contracts/utils/Context.sol

File Dependency Graph



Finding Summary



Manual Review Findings

ID	Title	Type	Severity	Resolved
IER-01M	Unlocked Compiler Version	Language Specific	Informational	✓
PTN-01M	Centralization concern	Volatile Code	Minor	✓
PTN-02M	Unlocked Compiler Version	Language Specific	Informational	✓
PTV-01M	Centralization concern	Volatile Code	Minor	✓
PTV-02M	Lack of input validation	Volatile Code	Minor	✓
PTV-03M	Unlocked Compiler Version	Language Specific	Informational	✓
PTV-04M	Contract not using Ownable.sol from OZ	Language Specific	Informational	✓
PTV-05M	sqrt function can be more gas efficient.	Gas Optimization	Informational	✓
PTV-06M	Order of arguments doesn't correspond to the function arguments	Inconsistency	Informational	✓
PTV-07M	Mark function as external	Language Specific	Informational	✓
SMH-01M	Unlocked Compiler Version	Language Specific	Informational	✓
CON-01M	Unlocked Compiler Version	Language Specific	Informational	✓

Static Analysis Findings

ID	Title	Type	Severity	Resolved
PTV-01S	Divide before multiply	Mathematical Operations	Medium	✓

IER-01M: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IERC20.sol L3

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 version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Polylastic development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

PTN-01M: Centralization concern

Type	Severity	Location
Volatile Code	Minor	PolylasticToken.sol L59

Description:

Owner of a token contract gets all of the total supply. In case of lost access to the private key of an account or mishandling security of private keys, an attacker could benefit from that.

Recommendation:

Total supply should be distirbuted to the users at start or be distributed to a handful of trusted addresses so no one address can hold all of the tokens at start. We would advise using multi-sig wallet for that.

Alleviation:

The Polylastic development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

PTN-02M: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	PolylasticToken.sol L3

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 version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Polylastic development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

PTV-01M: Centralization concern

Type	Severity	Location
Volatile Code	Minor	PolylasticTokenV2.sol L42, L58, L65, L76

Description:

Owner of a token contract gets all of the total supply. Owner also have all the power to set treasure address and whitelist wallets. In case of lost access to the private key of an account or mishandling security of private keys, an attacker could benefit from that.

Recommendation:

Total supply should be distirbuted to the users at start or be distributed to a handful of trusted addresses so no one address can hold all of the tokens at start.

We would advise using multi-sig wallet or Governance system.

Alleviation:

The Polylastic development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

PTV-02M: Lack of input validation

Type	Severity	Location
Volatile Code	Minor	PolylasticTokenV2.sol L58, L65, L76, L86

Description:

Linked functions lack an input validation against 0x0 address.

Recommendation:

We would advise to add input validation by having a require statement checking against 0x0 address.

Alleviation:

The Polylastic development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

PTV-03M: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	PolylasticTokenV2.sol L1

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 version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Polylastic development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

PTV-04M: Contract not using Ownable.sol from OZ

Type	Severity	Location
Language Specific	Informational	PolylasticTokenV2.sol L37, L48-L55, L59, L68, L79

Description:

Contract is not using Ownable.sol from OpenZeppelin which has well tested code with additional functionality like `onlyOwner` modifier which current contract is lacking. Functions like `transferOwnership` or `renounceOwnership` are also present and have proper events emitted.

Recommendation:

We would recommend to import and utilize Ownable.sol from OpenZeppelin instead of writing own implementation of said functionality. This will also help with simplifying the codebase.

Alleviation:

The Polylastic development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

PTV-05M: sqrt function can be more gas efficient.

Type	Severity	Location
Gas Optimization	Informational	PolylasticTokenV2.sol L134-L147

Description:

Current square root function for x==1, x==2, x==3 performs all the computation where in fact it could just return 1.

Recommendation:

We would recommend to use Babylonial method, utilized by Uniswap for computing square root.

```
    function sqrt(uint y) internal pure returns (uint z) {
            if (y > 3) {
                z = y;
                uint x = y / 2 + 1;
                while (x < z) {
                    z = x;
                    x = (y / x + x) / 2;
                }
            } else if (y != 0) {
                z = 1;
            }
        }
```

Alleviation:

Issue has been resolved. Recommendations were appliied.

PTV-06M: Order of arguments doesn't correspond to the function arguments

Type	Severity	Location
Inconsistency	Informational	PolylasticTokenV2.sol L150, L167

Description:

`computeTransferFee` function defines it's arguments in the following order `computeTransferFee(address sender, address receiver, uint amount)` . In the linked computeTransferFee() invocations, order is switched to `computeTransferFee(recipient, _msgSender(), amount);` whereas it should be `computeTransferFee(_msgSender(), recipient, amount);`

Recommendation:

Change the order of params in the functions invocations from `computeTransferFee(recipient, _msgSender(), amount);` to `computeTransferFee(_msgSender(), recipient, amount);`

Alleviation:

Issue partially resolves, only line 150 was fixed.

PTV-07M: Mark function as external

Type	Severity	Location
Language Specific	Informational	PolylasticTokenV2.sol L65, L76

Description:

Linked functions are not called from the contract they are defined so they could be marked as external. Additionaly `wallets` parameters of both functions can be also declared as `calldata`.

Recommendation:

Mark functions as external and make parameters `calldata`.

Alleviation:

The Polylastic development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

SMH-01M: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	SafeMath.sol L3

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 version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Polylastic development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

CON-01M: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Context.sol L3

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 version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The Polylastic development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

PTV-01S: Divide before multiply

Type	Severity	Location
Mathematical Operations	Medium	PolylasticTokenV2.sol L127

Description:

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

Recommendation:

Consider ordering multiplication before division.

Alleviation:

Issue has been resolved.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that 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.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

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`.

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.