PolymathNetwork

Security Assessment

February 4th, 2021

For :
PolymathNetwork

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Georgios Delkos @ CertiK
georgios.delkos@certik.io

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 indeed 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	PolymathNetwork
Description	A ethreum bridge to polymesh contract.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. da758815f71d4d5b49d7bd42bca64b0eeeaea685

Audit Summary

Delivery Date	February 4th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	January 29th, 2021 - January 30th, 2021

Vulnerability Summary

Total Issues	2
Total Critical	0
Total Major	0
Total Medium	0
Total Minor	1
Total Informational	1

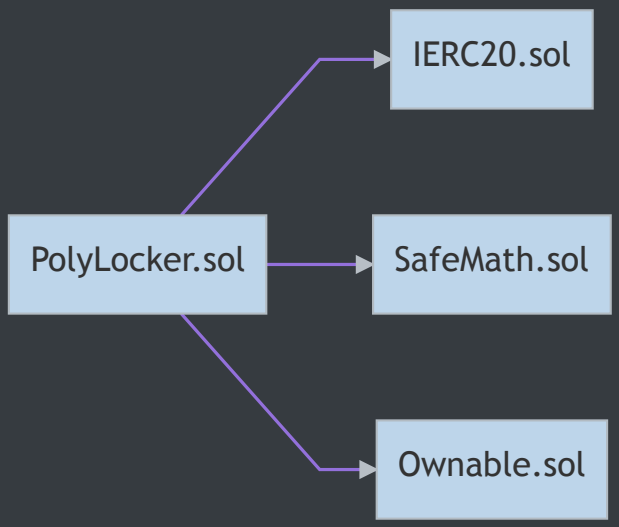
Executive Summary

Polymath requested for CertiK to perform an audit in their new protocol implementation. The auditing team conducted the audit in the timeframe between January 29, 2020, and January 30, 2021, with two engineers. The auditing process evaluated code implementation against provided specifications, examining language-specific issues, and performed manual examination of the code.
The code examined by the team did not have any significant or critical issues, and the team alleviated all problems presented by the auditing team.

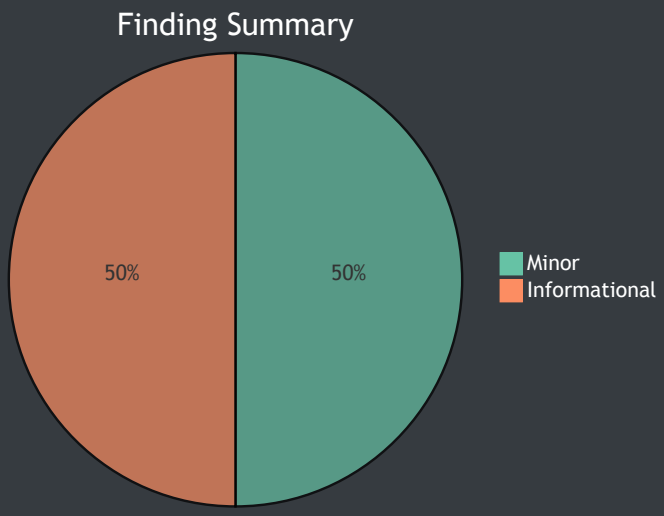
Files In Scope

ID	Contract	Location
PLR	PolyLocker.sol	contracts/PolyLocker.sol

File Dependency Graph (BETA)



Findings



ID	Title	Type	Severity	Resolved
PLR-01	Unlocked Compiler Version	Language Specific	Informational	✓
PLR-02	Logical Error	Logical Issue	Minor	✓

PLR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	PolyLocker.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:

Fixed by the team in commit 66f83b30b2a56da03e138a5b519f42361ad18ed7.\

PLR-02: Logical Error

Type	Severity	Location
Logical Issue	Minor	PolyLocker.sol L107

Description:

The comment states Make sure balance is divisible by 10e18 but the functionality does not represent that.

Recommendation:

Refactor the code using modulo to match the desired outcome.

Alleviation:

After coordinating with the Poly team, they clarified that the comment was actually incorrect and was not representing the desired functionality and thus they removed the misleading comment nullifying this exhibit.

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.

Logical Issue

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

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

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

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

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 and 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.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.