Brydge Security Analysis

by Pessimistic

This report is public

May 13, 2022

Abstract

In this report, we consider the security of smart contracts of Brydge project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of Brydge smart contracts. We performed our audit according to the procedure described below.

The initial audit showed one critical issue: Bug.
It also revealed one issue of medium severity: ERC20 standard violation. Moreover, several low-severity issues were found.

After the initial audit, the code base was updated. The developers fixed all of the issues and provided comments.

General recommendations

We recommend paying attention to the note on System design.

Project overview

Project description

For the audit, we were provided with Brydge project on a private GitHub repository, commit 1fc9893894c3401eb5c1a36d0c771febe75f8a0d.

The project has README.md file with a short description of the project, as well as detailed documentation. The code has some NatSpec comments.

All 3 tests pass, only 54.55% of branches of the most crucial contract are covered.

The total LOC of audited sources is 324.

Code base update

After the initial audit, the code base was updated. For the recheck we were provided with commit 92fa4efcc9d810e26eae04b8d940a7855c289d83.

In this update, all of the issues were fixed. Also, developers provided additional description and comments regarding the update.

The developers implemented new tests. As a result, all the 7 tests pass and the branch coverage is increased to 89%.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan the project's codebase with the automated tool Slither.
  - We manually verify (reject or confirm) all the issues found by the tools.
- Manual audit
  - We manually analyze the codebase for security vulnerabilities.
  - We assess the overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

C01. Bug in fee calculation (fixed)

In `PolygonPortal.sol` there is a bug in calculation in `mulDiv` function. The returned result should be the following: `a * c * z + a * d + b * c + (b * d) / z`.

The issue has been fixed and is not present in the latest version of the code.

Medium severity issues

Medium issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. ERC20 standard violation (fixed)

ERC-20 standard states:

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned!
```

However, in `PolygonPortal.sol` returned values from `transfer` calls are not checked.

The issue has been fixed and is not present in the latest version of the code.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Code quality (fixed)

There is a call to `beneficiary` address at line 56 in `PolygonPortal`. However, the call result is not checked. Despite the fact that `beneficiary` is a known address and calls should not revert, we still recommend checking their results since it is a standard pattern.

The issue has been fixed and is not present in the latest version of the code.

L02. Code quality (fixed)

We recommend marking constant addresses as immutable in `EthereumConstants.sol`.

The issue has been fixed and is not present in the latest version of the code.

L03. Function visibility (fixed)

Consider declaring `SwapERC20AndCall` function as external instead of public in order to improve code readability.

The issue has been fixed and is not present in the latest version of the code.

L04. Tests (fixed)

The current setup process for testing is too complex. We recommend describing the testing process in the documentation and reviewing required environment variables.

The issue has been fixed and is not present in the latest version of the code.

L05. Gas costs (fixed)

We recommend using uniswap implementation of `toAsciiString` function since it requires approximately 40000 less gas.

The issue has been fixed and is not present in the latest version of the code.

Notes

N01. System design

The current system implementation does not perform business-related security checks. For example, when swapping tokens it is a common practice to have a slippage check. This helps preventing unexpected token loss due to price fluctuations. However, in the `PolygonPortal` contract security checks are meant to be done by external routers and contracts called in `_handleCalls` function. Since these addresses are out of scope and since they are passed as arguments and can be changed at any time, there are no security guarantees.

The second caveat of the system is that it requires complex `msg.data` at layer 1. This means, that responsibility for data construction will probably be given to the off-chain code, which is out of scope. Since executed transactions will be complex and `PolygonPortal` is able to do arbitrary calls, the outcome of a bug or mistake cannot be predicted. Hence, we recommend adding explicit business-related security checks to the `PolygonPortal` contract even if all of the addresses and servers are trusted.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Pavel Kondratenkov, Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, Founder

May 13, 2022