SALUS SECURITY

MAR 2024

CODE SECURITY ASSESSMENT

POLYHEDRA

Overview

Project Summary

- Name: Polyhedra - zkBridgeV2
- Platform: EVM-compatible Chains
- Language: Solidity
- Repository:
    - https://github.com/zkBridge-integration/zkbridge-contracts-v2-optimized-audit
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Polyhedra - zkBridgeV2
Version	v2
Type	Solidity
Date	Mar 11 2024
Logs	Mar 08 2024; Mar 11 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	2
Total informational issues	1
Total	3

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Third-party dependencies	Low	Dependency	Acknowledged
2	Lack of upper limit for fees in setFee()	Low	Centralization	Acknowledged
3	Implementation can be initialized	Informational	Configuration	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Third-party dependencies

Severity: Low	Category: Dependency
Target: - contracts/ZKBridge.sol	

Description

contracts/ZKBridge.sol:L102

```
function validateTransactionProof(
    uint16 _srcChainId,
    bytes32 _srcBlockHash,
    uint256 _logIndex,
    bytes calldata _mptProof
) external {
    ...
    IMptVerifier.Receipt memory receipt = mptVerifier.validateMPT(_mptProof);
    require(blockUpdater.checkBlock(_srcBlockHash, receipt.receiptHash), "ZKBridge:Block
Header is not set");
    ...
}
```

The ZKBridge contract relies on the blockUpdater contract to enable permission checks for rootHash. The current audit treats third-party entities as black boxes and assumes they are working correctly. However, in reality, third parties could be compromised, resulting in the MPT validation of ZKBridge being under-validated or unavailable.

Recommendation

We understand that the business logic requires interaction with the third parties. We encourage the team to regularly monitor the statuses of third parties to reduce the impacts when they are not functioning properly.

Status

This issue has been acknowledged by the team. The team has stated that blockUpdater was not within the scope of this audit and has undergone thorough audits in the past.

2. Lack of upper limit for fees in setFee()

Severity: Low	Category: Centralization
Target: - contracts/ZKBridge.sol	

Description

The setFee() function is used to modify the value of the cross-chain fee.

contracts/ZKBridge.sol:L172-L178

```
function setFee(uint16 _dstChainId, uint256 _fee) public onlyFeeManager {
    fees[_dstChainId] = _fee;
    emit SetFee(_dstChainId, _fee);
}
```

However, the function has no upper limit on fees. If the fee setter's private key is compromised, the attacker can set cross-chain fee to type(uint256).max, preventing users from performing cross-chain operations.

Recommendation

It is recommended to include a reasonable upper limit for cross-chain fee in setFee().

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Implementation can be initialized

Severity: Informational	Category: Configuration
Target: - contracts/ZKBridge.sol	

Description

The ZKBridge contract uses the Initializable module from OpenZeppelin.
According to OpenZeppelin's documentation, it's best to invoke the `_disableInitializers` function in the constructor to prevent the implementation contract from being initialized by malicious users.

Recommendation

Consider using the _disableInitializers() function in the constructor to prevent malicious initialization of the Implement contract.

Status

The team has resolved this issue in commit 0fc4e20.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit bc64edf0:

File	SHA-1 hash
MptVerifier.sol	3be7d84737f086c97823117e8ddf2a68bc23e8bc
OptimizedTransparentUpgradeableProxy.sol	01474fa3057853150b439fb3b25fec5ce389e0d6
ZKBridge.sol	3467e9388a780fe6b512a7e6dd31689a70f97c68
ZkBridgeAdmin.sol	abaf7f9b196263e0b56caa262be21b3e8f4d88b0
RLPReader.sol	8027a599b1b749abdd5ac54cc768b2125abdef26