SALUS SECURITY

NOV 2023

CODE SECURITY ASSESSMENT

POLYHEDRA

Overview

Project Summary

- Name: Polyhedra - token-liquidity-bridge-V2
- Platform: EVM-compatible Chains
- Language: Solidity
- Repository: https://github.com/zkBridge-integration/token-liquidity-bridge-V2-audit
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Polyhedra - token-liquidity-bridge-V2
Version	v2
Type	Solidity
Date	Nov 27 2023
Logs	Nov 24 2023; Nov 27 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	4
Total informational issues	4
Total	8

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	The transfer on dstChain may fail	Low	Business Logic	Acknowledged
2	Third-party dependencies	Low	Dependency	Acknowledged
3	Missing event	Low	Logging	Acknowledged
4	Centralization risk	Low	Centralization	Mitigated
5	Incorrect event parameters	Informational	Logging	Acknowledged
6	Implementation can be initialized	Informational	Configuration	Acknowledged
7	Incorrect error message	Informational	Code Quality	Acknowledged
8	Floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

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

1. The transfer on dstChain may fail

Severity: Low	Category: Business Logic
Target: - src/Bridge.sol	

Description

a) Due to the need for separate transactions on both srcChain and dstChain in cross-chain transfers, the atomicity of the transaction is compromised. If the transfer on dstChain reverts, it will result in users being unable to access funds, and the funds will be locked in the Bridge.

Attach Scenario:

1. User initiates a cross-chain transfer of native token to a contract on dstChain without a fallback function. This transaction succeeds on srcChain.
2. Relayer initiates a transaction on dstChain to transfer funds to the same contract, but the lack of a fallback function causes the transaction to fail.

This results in the user's funds being transferred to the Bridge on srcChain without the proper refund logic.

b) This scenario can also occur when users transfer funds to the zero address, causing the transfer transaction on dstChain to fail due to ERC20 checks on the zero address

Recommendation

It is recommended that the refund logic be written in the contract rather than processed manually.

Status

This issue has been acknowledged by the team.

2. Third-party dependencies

Severity: Low	Category: Dependency
Target: - src/Bridge.sol	

Description

The Bridge contract relies on zkBridgeEndpoint and I1Bridge to enable seamless cross-chain transfers. 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 cross-chain transfer functionality of Bridge being 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.

3. Missing event

Severity: Low	Category: Logging
Target: - src/Bridge.sol	

Description

src\Bridge.sol:L127-L147

```
function transferETHMux(uint16 dstChainId, uint256 amount, address recipient) external
payable nonReentrant {
    if (
        _poolInfo[NATIVE_TOKEN_POOL_ID].balance + amount <=
_poolInfo[NATIVE_TOKEN_POOL_ID].maxLiquidity
            && amount <= _dstChains[NATIVE_TOKEN_POOL_ID][dstChainId].maxTransferLimit
    ) {
        require(msg.value >= amount, "Bridge: Insufficient ETH");
        uint256 fee = _transfer(dstChainId, NATIVE_TOKEN_POOL_ID, amount, recipient,
msg.value - amount);
        refundAmount = msg.value - amount - fee;
    } else {
        uint256 fee = l1Bridge.fees(dstChainId);
        require(msg.value >= amount + fee, "Bridge: Insufficient ETH");
        l1Bridge.transferETH{value: amount + fee}(dstChainId, amount, recipient);
        refundAmount = msg.value - amount - fee;
    }
}
```

The transferETHMux() function emits a TransferToken event when the logic enters the if clause. However, the TransferToken event is missing when the logic enters the else clause. The same issue also applies to the transferTokenMux function.

Recommendation

It is recommended to add the missing event.

Status

This issue has been acknowledged by the team.

4. Centralization risk

Severity: Low	Category: Centralization
Target: - src/Pool.sol	

Description

There exists a function removeLiquidity in Pool controlled by PoolManager. If the PoolManager's private key is compromised, the attacker can exploit the removeLiquidity function to withdraw all the tokens from the bridge.

src\Pool.sol:L139-L150

```
function removeLiquidity(uint256 poolId, uint256 amount) external onlyPoolManager
nonReentrant {
    _checkPool(poolId);
    _checkConvertRate(poolId, amount);
    require(amount <= _poolInfo[poolId].balance);
    if (poolId == NATIVE_TOKEN_POOL_ID) {
        Address.sendValue(payable(msg.sender), amount);
    } else {
        IERC20(_poolInfo[poolId].token).safeTransfer(msg.sender, amount);
    }
    _poolInfo[poolId].balance -= amount;
    emit RemoveLiquidity(poolId, amount);
}
```

Recommendation

We recommend transferring privileged accounts to multi-sig accounts for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

The team has mitigated this issue by configuring the privileged accounts as multi-sig addresses.

2.3 Informational Findings

5. Incorrect event parameters

Severity: Informational	Category: Logging
Target: - src/Bridge.sol	

Description

The events triggered in _transfer and zkReceive both use the converted data `amountSD`. However, it is more suitable to use the unconverted `amount` variable in the events.

src\Bridge.sol:L66-L86

```
function _transfer(uint16 dstChainId, uint256 poolId, uint256 amount, address recipient,
uint256 fee) internal returns (uint256) {
    ...
    uint256 amountSD = _deposit(poolId, dstChainId, amount);
    emit TransferToken(sequence, dstChainId, poolId, msg.sender, recipient, amountSD);
    ...
}
```

src\Bridge.sol:L106-L124

```
function zkReceive(uint16 srcChainId, address srcAddress, uint64 sequence, bytes
calldata payload) external nonReentrant {
    ...
    (uint256 poolId, uint256 amountSD, address recipient) = abi.decode(payload,
(uint256, uint256, address));
    uint256 amount = _withdraw(poolId, srcChainId, amountSD);

    emit ReceiveToken(sequence, srcChainId, poolId, recipient, amountSD);
    ...
}
```

Recommendation

It is recommended to use the `amount` value instead of `amountSD` in the said events.

Status

This issue has been acknowledged by the team.

6. Implementation can be initialized

Severity: Informational	Category: Configuration
Target: - src/Bridge.sol	

Description

The Bridge 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

This issue has been acknowledged by the team.

7. Incorrect error message

Severity: Informational	Category: Code Quality
Target: - src/Pool.sol	

Description

src\Pool.sol:L200-L203

```
function _deposit(uint256 poolId, uint16 dstChainId, uint256 amount) internal returns
(uint256) {
    ...
    require(
        _poolInfo[poolId].balance + amount <= _poolInfo[poolId].maxLiquidity,
        "Pool: Insufficient liquidity on the target chain"
    );
    ...
}
```

The highlighted error message does not match the code logic.

This code is intended to ensure that the liquidity of a certain token does not exceed the limit. However, the error message refers to the liquidity of the target chain, which does not align with the intended logic.

Recommendation

It is recommended to modify the error message to align with the code for consistency..

Status

This issue has been acknowledged by the team.

8. Floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.0;
```

The project is using a floating solidity version.

Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively or recently released pragma versions may have unknown security vulnerabilities.

Recommendation

Consider locking the pragma in all the contracts.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit a3a5b8a:

File	SHA-1 hash
Admin.sol	620adb8b39bbd09fff2497e78d4861f1809dda3e
Bridge.sol	2f961efca5930bcec3ec659acc25bf1662df3b55
Pool.sol	6ec204ba7e009cb19505767ebc6386cf6102ad2d
ZkBridgeAdmin.sol	abaf7f9b196263e0b56caa262be21b3e8f4d88b0
IL1Bridge.sol	1fbaa23d2d65b7831c1f0e809e439890c2dfde27
IZKBridgeEndpoint.sol	8a8b581666a8c5fca7f62f8b2dc5baff9bf3f23a
IZKBridgeReceiver.sol	6119390d20574b0d85e005ce4267fe6ecaad7b7b