OOOO Bridge
Preliminary Audit Report

Mon Apr 01 2024

contact@scalebit.xyz https://twitter.com/scalebit_

ScaleBit

OOOO Bridge Preliminary Audit Report

1 Executive Summary

1.1 Project Information

Description	A protocol for batch ETH transfers
Type	Asset Management
Auditors	ScaleBit
Timeline	Mon Apr 01 2024 - Mon Apr 01 2024
Languages	Solidity
Platform	Ethereum
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/l2-bridge/oooo-bridge-contracts
Commits	9281e29b490b892e1b70a78f77f0796109afcf69

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
BTR	contracts/BatchTransfer.sol	e7bfeebba233c7f1dc421b458ddbcf9e3ef8e250

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	0	0
Informational	0	0	0
Minor	1	0	0
Medium	1	0	0
Major	1	0	0
Critical	0	0	0
Discussion	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by OOOO to identify any potential issues and vulnerabilities in the source code of the OOOO Bridge smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 3 issues of varying severity, listed below.

ID	Title	Severity	Status
BTR-1	Single-step Ownership Transfer Can be Dangerous	Major	Pending
BTR-2	`call()` Should be Used Instead of `transfer()` on An Address Payable	Medium	Pending
BTR-3	Lack of Events Emit	Minor	Pending

3 Participant Process

Here are the relevant actors with their respective abilities within the OOOO Bridge Smart Contract :

Admin

- The admin can perform batch transfers of ETH using the `batchTransfer` function.
- The admin can withdraw the contract's balance using the `withdraw` function.

4 Findings

BTR-1 Single-step Ownership Transfer Can be Dangerous

Severity: Major

Status: Pending

Code Location:

contracts/BatchTransfer.sol#7

Descriptions:

Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. If the admin permissions are given to the wrong address within this function, it will cause irreparable damage to the contract.

```
// Importing OpenZeppelin's Ownable contract to manage ownership
import "@openzeppelin/contracts/access/Ownable.sol";
```

Suggestion:

It is recommended to use a two-step ownership transfer pattern, meaning ownership transfer gets to a "pending" state and the new owner should claim his new rights, otherwise, the old owner still has control of the contract.

BTR-2 `call()` Should be Used Instead of `transfer()` on An Address Payable

Severity: Medium

Status: Pending

Code Location:

contracts/BatchTransfer.sol#36

Descriptions:

The `transfer()` and `send()` functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction.

```
function withdraw() external onlyOwner {
      payable(owner()).transfer(address(this).balance);
  }
```

Suggestion:

It is recommended to use `call()` instead of `transfer()` .

BTR-3 Lack of Events Emit

Severity: Minor

Status: Pending

Code Location:

contracts/BatchTransfer.sol#14-39

Descriptions:

The smart contract lacks appropriate events for monitoring sensitive operations, which could make it difficult to track important actions or detect potential issues. So it is recommended to add relevant events for some important functions like `batchTransfer`, `withdraw`.

Suggestion:

It is recommended to emit events for those important functions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer