Governance Crosschain Bridges Smart Contracts Security Audit Report

August 15, 2022

0XORIO

List of contents

1 Introduction

1.1 Disclaimer

The audit makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about the fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors is involved in the work on this audit. Each of them checks the provided source code independently of each other in accordance with the security assessment methodology described below:

1. Project architecture review:

Manually code study of the architecture of the code based on the source code only to find out the errors and bugs.

2. Check the code against the list of known vulnerabilities

The verification process of the code against the constantly updated list of already known vulnerabilities maintained by the company.

3. Architecture and structure check of the security model

Study project documentation and its comparison against the code, including the study of the comments and other technical papers.

4. Result's cross-check by different auditors

Normally the research of the project is made by more than two auditors. After that, there is a step of the mutual cross-check process of audit results between different task performers.

5. Report consolidation

Consolidation of the audited report from multiple auditors.

6. Reaudit of new editions

After the client's review and fixes, the found issues are double-checked. The results are provided in the new audit version.

7. Final audit report publication

The final audit version is prepared and provided to the client and also published on the official website of the company.

1.2.1 Severity Level Reference

Findings discovered during the audit are classified as follows: Every issue in this report was assigned a severity level from the following:

- **CRITICAL**: A bug leading to assets theft, fund access locking, or any other loss of funds due to transfer to unauthorized parties.
- **MAJOR**: A bug that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
- **WARNING**: A bug that can break the intended contract logic or expose it to DDoS attacks.
- **INFO**: Minor issue or recommendation reported to / acknowledged by the client's team.

1.2.2 Status Level Reference

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

- **NEW**: Waiting for the project team's feedback.
- **FIXED**: Recommended fixes have been made to the project code, and the identified issue no longer affects the project's security.
- **ACKNOWLEDGED**: The project team is aware of this finding. Recommended fixes for this finding are planned to be made. This finding does not affect the overall security of the project.
- **NO ISSUE**: Finding does not affect the overall security of the project and does not violate the logic of its work
- **DISMISSED**: The issue or recommendation was dismissed by the client.

1.3 Project overview

This scope of the contracts contains the crosschain governance bridges to be used by the Lido Governance to execute the governance proposals across different networks.

1.4 Audit Scope

The scope of the audit includes the following smart contracts at: OptimismBridgeExecutor.sol ArbitrumBridgeExecutor.sol AddressAliasHelper.sol L2BridgeExecutor.sol BridgeExecutorBase.sol

The audited commit identifier is [8fa25b0080dd3dcc2390313631aea6796a12c9d8]

2 Report

2.1 CRITICAL

No critical issues found

2.2 MAJOR

No major issues found

2.3 WARNING

2.3.1 Possible to execute expired actionSet in the future

Severity	WARNING
Status	ACKNOWLEDGED

Description

If the government change `gracePeriod` to a bigger one it'll be possible to execute some expired actions BridgeExecutorBase.sol#L231

```
} else if (block.timestamp > actionsSet.executionTime +
_gracePeriod) {
    return ActionsSetState.Expired;
} else {
    return ActionsSetState.Queued;
}
```

And guardian can't cancel them because canceling expired actions is disallowed BridgeExecutorBase.sol#L110-L111

```
function cancel(uint256 actionsSetId) external override onlyGuardian
{
    if (getCurrentState(actionsSetId) != ActionsSetState.Queued)
revert OnlyQueuedActions();
```

It can be a threat if one of contracts that was in the actionsSet is compromised, so an owner of that contract can execute any code and change any storage slot (if that was delegatecall), steal some funds (if value was not 0). It may also call selfdestruct

The worst case is that in several years the government decides to increase gracePeriod for very long time.
And very old `Expired` actionSet becomes `Queued`.
And there is a proposal to `delegatecall` to some contract that is now controlled by an untrusted entity. It may be unexpected because that action was considered expired. And even if it's expected there will be a choice when we either increase gracePeriod and try to cancel that action before the attacker use it or don't update gracePeriod at all.

Recommendation

Consider allowing to cancel expired actions or make it clear in the docs that expired actions can become queued in the future. Another option may be to add expiresAt field to the ActionSet so an expired action can not become queued.

2.3.2 Guardian may cancel `updateGuardian`

Severity	WARNING
Status	ACKNOWLEDGED

Description

A guardian may block guardian updates so it's impossible to change guardian without its permission which may not be desired (government should be higher in rank than a network's guardian).
So if a guardian is compromised and start misbehaving (cancel every or some proposals) there is no way to change it.

Recommendation

Consider adding a way to update a guardian without its permission

2.4 INFO

2.4.1 Execution ordering may be unexpected

Severity	INFO
Status	ACKNOWLEDGED

Description

It's possible that actionSets are executed in an order that is not the same, as they were accepted by the government on Ethereum. Because `execute` function does not check the order and the gap is not big enough it's possible to execute second proposal first for different reason, e.g. gas price on a first proposal was set too low and it's execution is postponed. The order also may be different because of bridges that relay messages. If the first message reverts (e.g. maxSubmissionCost is low in Arbitrum) we will need to retry it by hand. While we do it the second message may come through.

It may be unexpected in some cases.

It can be bad if we change some critical parameters in the second proposal, e.g. we want to make a set of actions in the first proposal and then in the second proposal call `updateEthereumGovernanceExecutor`. If the second proposal is executed first, the first proposal will revert on `queue` because of `onlyEthereumGovernanceExecutor` modifier.

Another example is when in the second proposal gracePeriod is lowered. So first proposal will be Expired.

Recommendation

Consider adding this information to docs. If order matters government should wait while the first proposal is queued and have enough gap between them so `execute` is called by someone.

2.4.2 Changing `gracePeriod` may change actionSets' statuses

Severity	INFO
Status	ACKNOWLEDGED

Description

It may be unexpected that setting `gracePeriod` may change statuses of old issues or block execution of queued actions.
Some Queued actions may revert before some other condition is met. So it's desired to keep them Queued until it stops to revert. But proposal that decreases `gracePeriod` may make that action `Expired`.

Recommendation

It should be mentioned in the docs. Another option may be to add expiresAt field to the ActionSet so an `Expired` action can not become `Queued`. And `Queued` can't become `Expired`.

2.4.3 `delegatecall` may be unnecessary

Severity	INFO
Status	ACKNOWLEDGED

Description

Allowing `delegatecall` allows government to corrupt the storage of the contract or change storage variables without emitting an event.
We also need to fully trust the target (especially considering gracePeriod increase, see Warning#1).
But it gives additional advantages that makes a contract and its storage extendable.

Recommendation

Depending on your use case you may want to narrow the attack or error surface by removing it.

2.4.4 Denying duplicate actions may be redundant

Severity	INFO
Status	ACKNOWLEDGED

Description

It's not clear why duplicate actions in the same proposal are forbidden. BridgeExecutorBase.sol#L300-L310

```
bytes32 actionHash = keccak256(
abi.encode(
    targets[i],
    values[i],
    signatures[i],
    calldatas[i],
    executionTime,
    withDelegatecalls[i]
)
);
if (isActionQueued(actionHash)) revert DuplicateAction();
```

It can be easily circumvented by including signature in calldata BridgeExecutorBase.sol#L352-L356

```
if (bytes(signature).length == 0) {
    callData = data;
} else {
    callData = abi.encodePacked(bytes4(keccak256(bytes(signature))),
data);
}
```

or by setting a different signature but with the same `bytes4(keccak256(bytes(signature))` hash.

Or by just moving the duplicated action in a different proposal. So executionTime is different.

It's also possible even so highly unlikely to get the same hash in case when we have to identical actions and `_delay` was changed between proposals to a lower one so `executionTime` will be the same and `queue` of an otherwise valid ActionSet will revert. But it's negligible.

And checking and changing `_queuedActions[actionHash]` in `queue`, `execute`, and `cancel` consume gas.

Recommendation

Consider removing `isActionQueued` checks.
Consider removing `signatures` array and only using `calldatas` to store a signature hash so duplicated actions are not possible if this is desired behavior.

2.4.5 A struct can be introduced to improve readability

Severity	INFO
Status	ACKNOWLEDGED

Description

Multiple arrays in `queue`

```
  function queue(
    address[] memory targets,
    uint256[] memory values,
    string[] memory signatures,
    bytes[] memory calldatas,
    bool[] memory withDelegatecalls
  ) external onlyEthereumGovernanceExecutor {
    _queue(targets, values, signatures, calldatas,
```

```
withDelegatecalls);
  }
```

`_queue` and other functions may be replaced with an array of structs.

Benefits:

- More readable code. Instead of list of parameters, you'll get an array of structs
- Reduce duplication when the same arrays passed to several functions
- No need to check length of every array in `queue`

```
struct Action {
  address target;
  uint256 value;
  string signature;
  bytes aCalldata;
  bool withDelegatecall;
}
...
function queue(Action[] actions)
```

Recommendation

Consider replacing several arrays with an array of structs

2.4.6 Queue and deposit requires several government actions

Severity	INFO
Status	ACKNOWLEDGED

Description

`queue` function is not payable. Queueing ActionSets that need some value requires several government actions. Deposit and queue. It makes a government action not atomic.

Recommendation

Consider allowing to submit msg.value to `queue` by making in `payable`.

3 Conclusion

The following table contains the total number of issues that were found during audit:

Level	Amount
CRITICAL	0
MAJOR	0
WARNING	2
INFO	6
Total	8

Smart contracts have been audited and no critical or major issues were found. Also several recommendations were marked as warning and informational. Some changes were proposed to follow best practices, reduce potential attack surface, simplify code maintenance and increase its readability.

All the suggestions were marked as acknowledged by the team. As no found issues have major or critical severity, contracts are assumed to be secure to use according to our security criteria and ready to deploy to mainnet.

4 About Oxorio

Oxorio is a young but rapidly growing audit and consulting company in the field of the blockchain industry, providing consulting and security audits for organizations from all over the world. Oxorio has participated in multiple blockchain projects where smart contract systems were designed and deployed by the company.

Oxorio is the creator, maintainer, and major contributor of several blockchain projects and employs more than 5 blockchain specialists to analyze and develop smart contracts.

Contacts:

- oxor.io
- ping@oxor.io
- github
- linkedin