Alethea AI LayerZero ERC20 Tunnel for Ethereum <-> BNB <-> opBNB bridging support

Smart Contract Pre-Audit Check and Code Review

Version 1

Smart Contract Pre-Audit Check and Code Review

Prepared by: Darren Jensen

Version: 1
Date: 22nd August 2023

Introduction

This document outlines the findings for smart contract code review for contracts in ai-protocol-contracts repo at commit SHA `e385a911`. This report covers all contracts under the contracts/layer_zero directory as well as tests under test/layer_zero directory and deployment scripts under deploy/v2_8 directory.

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behaviour with some of the protocol's functionalities that's not so critical.

Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions and the cost of the attack is relatively low to the amount of funds that can be stolen or lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - has too many or too unlikely assumptions or requires a huge stake by the attacker with little or no incentive.

Severity classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Low	Low	Low

Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.
- **Informational** - client **could** consider design/UX related decision
- **Recommendation** - client **could** have an internal team discussion on whether the recommendations provide any UX or security enhancement and if it is technically and economically feasible to implement the recommendations
- **Gas Findings** - client **could** consider implementing suggestions for better UX

Overview

Project Name	Alethea AI LayerZero ERC20 Tunnel
Repository	ai-protocol-contracts
Commit SHA	`e385a911`
Documentation	Provided
Methods	Manual review & CLI review (Mythril, Slither, Solhint)

Contracts in Scope

This “Version 1” report reviews all contracts under the `contracts/layer_zero` directory in the ai-protocol-contracts repository.

LzERC20ChildTunnelV1.sol
LzERC20RootTunnelV1.sol

The report also focuses on the associated tests and Hardhat deployment scripts for these contracts.

Issues found

Severity	Count
High risk	1
Medium risk	0

Low risk	7
Informational	0
Recommendations	0
Gas Findings	0

High Findings

[H-1] Unpacking order of the `srcAddresses` in `__lzReceive` for both root and child tunnel contracts appears to be reversed.

Context

https://github.com/AletheaAI/ai-protocol-contracts/blob/e385a91161108f5468e3ba646a48e33e3204e646/contracts/layer_zero/LzERC20RootTunnelV1.sol#L265 **(lzReceive in Root)**

https://github.com/AletheaAI/ai-protocol-contracts/blob/e385a91161108f5468e3ba646a48e33e3204e646/contracts/layer_zero/LzERC20ChildTunnelV1.sol#L247 **(lzReceive in Child)**

https://github.com/AletheaAI/ai-protocol-contracts/blob/e385a91161108f5468e3ba646a48e33e3204e646/contracts/layer_zero/LzERC20RootTunnelV1.sol#L432 **(sendMessageToChild in Root)**

https://github.com/AletheaAI/ai-protocol-contracts/blob/e385a91161108f5468e3ba646a48e33e3204e646/contracts/layer_zero/LzERC20ChildTunnelV1.sol#L403 **(sendMessageToRoot in Child)**

Description

The `sendMessageToChild` function (in root) packs the `childTunnelAddress` followed by the `rootTunnelAddress`.

However in the `lzReceive` function (in child) the addresses are unpacked and assigns the variables in the opposite order - by unpacking the `rootTunnelAddress` then the `childTunnelAddress`. This would result in the `_rootTunnel` variable being set with the `childTunnelAddress`. Since there are `require` statements immediately following the variable unpacking this would result in a revert.

It is the same issue going in the opposite direction. The child packs the `rootTunnelAddress` then the `childTunnelAddress` and the root unpacks and assigns the `childTunnelAddress` first and then the `rootTunneladdress`.

The reason this issue is not detected in the tests is because the tests always pack the addresses in the expected order and then call the `lzReceive` function directly in the tunnel contract.

Recommended Mitigation Steps

Fix the unpacking order and assignment of tunnel address variables in the `__lzReceive` functions of both the root and the child tunnel contracts.

Low Findings

[L-1] Can encode an empty string for `src_address` data and the test will still pass.

Context

https://github.com/AletheaAI/ai-protocol-contracts/blob/e385a91161108f5468e3ba646a48e33e3204e646/test/layer_zero/lz_erc20_root_tunnel.js#L272

Description

The src_addresses can be defined as `const src_address = web3.utils.encodePacked("");` and the test will still pass.

Recommended Mitigation Steps

Update the `src_address` to an encoded empty string so its clear that the values have no impact on this particular test.

[L-2] `root_tunnel` and `child_tunnel` are defined incorrectly in test

Context

https://github.com/AletheaAI/ai-protocol-contracts/blob/e385a91161108f5468e3ba646a48e33e3204e646/test/layer_zero/lz_erc20_root_tunnel.js#L286

Description

The variables are assigned as follows:

```
root_tunnel = child_tunnel_address
child_tunnel = tunnel.address
```

Recommended Mitigation Steps

Switch the variable names to match the values being set such that the `fails` function signature is as follows:

```
async function fails(
      promise,
      error,
      chainId = child_chain_id,
      child_tunnel = child_tunnel_address,
      root_tunnel = tunnel.address,
      srcAddress = web3.utils.encodePacked(
            {value: child_tunnel, type: "address"},
            {value: root_tunnel, type: "address"}
      ),
)
```

[L-3] Comment *“child token gets minted"* should be *"child token gets burnt"* since it is apart of a withdrawal test

Context

https://github.com/AletheaAI/ai-protocol-contracts/blob/e385a91161108f5468e3ba646a48e33e3204e646/test/layer_zero/lz_erc20_root_tunnel.js#L317

Description

The comment refers to minting when the test is for burning.

Recommended Mitigation Steps

Update comment to reflect the test being performed.

[L-4] The test *'sender balance decreases as expected'* only checks for a zero balance.

Context

https://github.com/AletheaAI/ai-protocol-contracts/blob/e385a91161108f5468e3ba646a48e33e3204e646/test/layer_zero/lz_erc20_root_tunnel.js#L469

Description

The test 'sender balance decreases as expected' only checks for a zero balance. It should check that the balance decreased by a specific amount to avoid a potential false positive.

Recommended Mitigation Steps

Check for a specific change in balance or start with a higher balance and check a specific non zero balance after the test run..

[L-5] The test *'MessageSent'* event is emitted by LZ endpoint' only checks the values are echoed

Context

https://github.com/AletheaAI/ai-protocol-contracts/blob/e385a91161108f5468e3ba646a48e33e3204e646/test/layer_zero/lz_erc20_root_tunnel.js#L450

Description

The test *'MessageSent event is emitted by LZ endpoint (LzEndpoint)'* only checks if the values are echoed from the Mock endpoint.

Recommended Mitigation Steps

Ideally, there should also be a test that the Child tunnel receives the expected message and consumes it without error (integration test).

[L-6] Potential for using shared behaviours in tests to avoid duplication

Context

https://github.com/mochajs/mocha/wiki/Shared-Behaviours

Description

There are similarities between the functionality of the root and child tunnel tests such that shared behaviours (or similar approach) may be beneficial.

Recommended Mitigation Steps

Review the possibility of utilizing a shared behaviours (or similar) pattern for specification code reuse.

[L-7] Since the bridging is between two L1 networks (Ethereum <-> BNB) and then to a L2 network opBNB it is potentially confusing to refer to BNB as 'L2' and opBNB as 'L3'.

Context

https://github.com/AletheaAI/ai-protocol-contracts/blob/e385a91161108f5468e3ba646a48e33e3204e646/contracts/token/OpBnbAliERC20v2.sol#L13

Description

For clarity it might be better to use different terms to refer to the chains being bridged between, however I do understand that it is a 3 layer approach from the perspective of Alethea AI.

Recommended Mitigation Steps

Consider a different naming approach if deemed appropriate.

Contract versions used in audit

This report was conducted by using the contracts in SHA `e385a911`. Please note that the author ***did not make any modifications to the Smart Contracts***. All the SHA-256 smart contract file fingerprints are shown in Appendix A of this document and can be recalculated if needed to ensure the validity and expected code version of the contracts.

Test Run

The auditor built the contracts using hardhat and ran all the tests which are passing

All the contracts in the scope of this audit were compiled and the test run executed successfully with **all tests passing**.

Test Coverage

There is very good test coverage for both LayerZero Tunnel contracts:

```
------------------------------|----------|----------|----------|----------|----------------|
File                          |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
------------------------------|----------|----------|----------|----------|----------------|
 layer_zero/                  |    96.15 |    97.62 |      100 |    96.23 |                |
  LzERC20ChildTunnelV1.sol    |       96 |     97.5 |      100 |    96.08 |        223,413 |
  LzERC20RootTunnelV1.sol     |     96.3 |    97.73 |      100 |    96.36 |        241,442 |
------------------------------|----------|----------|----------|----------|----------------|
```

Linting

Linting is a valuable tool for finding potential issues in smart contract code. It can find stylistic errors, violations of programming conventions, and unsafe constructs in your code. There are many great linters available, such as Solhint. Linting can help find potential problems, even security problems such as re-entrancy vulnerabilities before they become costly mistakes.

After installation, Solhint can be run via the terminal as follows.

```
solhint 'contracts/*.sol'
```

This command will run `solhint` for the contracts in the root of the `contracts/layer_zero` directory. I have run the above command and included the output in the Appendix section of this report. The only issue identified by `solhint` was the length of the line in some instances. The full report is shown in the Appendix section of this report.

I recommend running the `solhint` linter, either via a Git commit hook or as part of an integration with the developer IDE so that the recommendations can be checked on every code change.

NOTE: the above issues are low priority and would not have any impact if not modified.

Slither Security Analysis

Auditor uses Slither Docker Image provided by TailOfBits for running Slither security analysis for all contracts. Since Docker is used there are two steps required - one to start up the Trail Of Bits container running locally and the other is to run the analysis command within the running container.

```
docker run -it -v $(pwd):/share trailofbits/eth-security-toolbox
```

Run the following command in the container:

```
slither --checklist --exclude-informational --exclude-low
--solc-remaps "@openzeppelin/=/share/node_modules/@openzeppelin/
../utils/=/share/contracts/utils/
../interfaces/=/share/contracts/interfaces/"
/share/contracts/layer_zero
```

The final report is available in the Appendix here.

Mythril Security Analysis

Auditor uses MythX for running Mythril security analysis for all contracts. The command to submit all token and protocol contracts to MythX using standard mode for analysis is:

```
mythx analyze --mode standard --remap-import
"@openzeppelin/=$(pwd)/node_modules/@openzeppelin/"
contracts/layer_zero/*.sol
```

The output shown in the Appendix is for a quick summary only. The main observations are that there are **no severities reported that are above the "Low" level of severity**. Most are either false positives or can be fixed very easily with minimal impact.

NOTE: Each contract has its own individual report (that can be downloaded as PDF) for each submitted contract to the MythX service. These reports are included in a separate ZIP file along with this main report.

Conclusion

The contracts are well written and well tested. The main issue to check is that of the packing and unpacking of addresses in the root and child tunnels and to check this functionality is operational in testnet and via an integration. There are some suggestions around the testing such as using shared behaviours to improve specification reuse. There were also no obvious issues with the deployment scripts.

Appendix A

LoC (Lines of Code)

The [cloc utility]was used to determine the lines of code under review. The utility excludes empty lines and comments to leave a count of auditable lines of code in each contract. Since all contracts in scope are under the `contracts/layer_zero` folder the `cloc` command was run once with the output as follows:

```
cloc contracts/layer_zero
       2 text files.
       2 unique files.
       0 files ignored.

github.com/AlDanial/cloc v 1.96  T=0.01 s (372.6 files/s, 197688.1 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Solidity                         2            110            698            253
-------------------------------------------------------------------------------
SUM:                             2            110            698            253
-------------------------------------------------------------------------------
```

SHA-256 File Fingerprints

The command to run to generate the SHA-256 fingerprint for the contracts under the `contracts/layer_zero` folder is:

```
shasum -a 256 contracts/layer_zero/*.sol
```

Which output the following SHA fingerprints for the two contracts under review:

```
3fbf7144aa55ba3204865daa924baff49caab2c5a488166536ab9c18ef051f62  LzERC20ChildTunnelV1.sol
c1dfd7336d4d4a903f97f1fa15ce40b705c3dc7cb71b494a0de5e8158ee91241  LzERC20RootTunnelV1.sol
```

Solhint Report

The command to `solhint` for all LayerZero tunnel contracts is as follows:

```
solhint contracts/layer_zero/*.sol
```

```
contracts/layer_zero/LzERC20ChildTunnelV1.sol
  207:2  error  Line length must be no more than 120 but current length is 125
max-line-length

contracts/layer_zero/LzERC20RootTunnelV1.sol
  225:2  error  Line length must be no more than 120 but current length is 125
max-line-length
```

✖ 2 problems (2 errors, 0 warnings)

Mythril Report

Below is the console output from the Mythril Report. Each report is available as a PDF file which will be included in a separate ZIP file along with this report. The output below is for a quick summary only. Main observations are that there are no severities reported that are above the “Low” level of severity. Most are either false positives or can be fixed very easily with minimal impact.

```
Report for contracts/layer_zero/LzERC20ChildTunnelV1.sol
https://dashboard.mythx.io/#/console/analyses/aa7be554-8712-4606-be83-6f274d34042f
```

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

```
Report for contracts/layer_zero/LzERC20RootTunnelV1.sol
https://dashboard.mythx.io/#/console/analyses/030e4d9d-f6d3-49e3-9246-af8048d73a43
```

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

Slither Report

Below are the Slither Report summary output for the layer zero contracts. The full markdown files are shared separately.

```
arbitrary-send (2 results) (High)
controlled-delegatecall (2 results) (High)
shadowing-state (2 results) (High)
unchecked-transfer (2 results) (High)
uninitialized-local (4 results) (Medium)
external-function (25 results) (Optimization)
```

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held understanding of the auditor's knowledge of security patterns as they relate to the client's contract(s), assuming that blockchain technologies, in particular, will continue to undergo frequent and ongoing development and therefore introduce unknown technical risks and flaws. The scope of the audit presented here is limited to the issues identified in the preliminary section and discussed in more detail in subsequent sections. The audit report does not address or provide opinions on any security aspects of the Solidity compiler, the tools used in the development of the contracts or the blockchain technologies themselves, or any issues not specifically addressed in this audit report.

The audit report 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, the legal framework for the business model, or any other statements about the suitability of the contracts for a particular purpose, or their bug-free status.

To the full extent permissible by applicable law, the auditors disclaim all warranties, express or implied. The information in this report is provided "as is" without warranty, representation, or guarantee of any kind, including the accuracy of the information provided. The auditors hereby disclaim, and each client or user of this audit report hereby waives, releases and holds all auditors harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.