# LayerZero

# Audit

Presented by:

**OtterSec** contact@osec.io

**Robert Chen** notdeghost@osec.io

**Naveen Kumar J** naina@osec.io

# Contents

**01 Executive Summary** **2**

Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

Key Findings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

**02 Scope** **3**

**03 Findings** **4**

**04 Vulnerabilities** **5**

OS-LZR-ADV-00 [high] [resolved] | Freeze Bridge with Invalid Sender . . . . . . . . . . . . . . . . 6

**05 General Findings** **8**

OS-LZR-SUG-00 | Inconsistent Config Parameter Checking . . . . . . . . . . . . . . . . . . . . . . 9

OS-LZR-SUG-01 | Inconsistent Table Error Handling . . . . . . . . . . . . . . . . . . . . . . . . . 10

OS-LZR-SUG-02 | Sanity Check Oracle Hash Size . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

OS-LZR-SUG-03 | Missing Address Size Checks . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

**06 Formal Verification** **13**

OS-LZR-VER-00 | Bridge Specifications . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

OS-LZR-VER-01 | Data Invariant Specifications . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

OS-LZR-VER-02 | Miscellaneous Function Specifications . . . . . . . . . . . . . . . . . . . . . . 16

# 01 | Executive Summary

## Overview

LayerZero engaged OtterSec to perform an assessment of the `layerzero` program. This assessment was conducted between August 29th and September 16th, 2022.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team over to streamline patches and confirm remediation. We delivered the final confirmation of the patches September 30th, 2022.

After the initial audit, we continued to work with the LayerZero team to review code changes.

## Key Findings

Over the course of this audit engagement, we produced 8 findings total.

In particular, we identified an issue (OS-LZR-ADV-00) that would allow a malicious user to freeze the token bridge. In order to remediate this issue, we worked with LayerZero to redesign the underlying endpoint architecture, introducing tighter segmentation on the messages.

We also made recommendations around formalizing address sizing, sanity-checking hash lengths, and example specifications to help with formal verification of LayerZero contracts.

# 02 | Scope

The source code was delivered to us in a git repository at github.com/LayerZero-Labs/LayerZero-Aptos-Contract. This audit was performed against commit `43cf4e1`. We then performed additional code reviews up to commit `7a96e9c`.

There were a total of two programs included in this audit. A brief description of the programs is as follows.

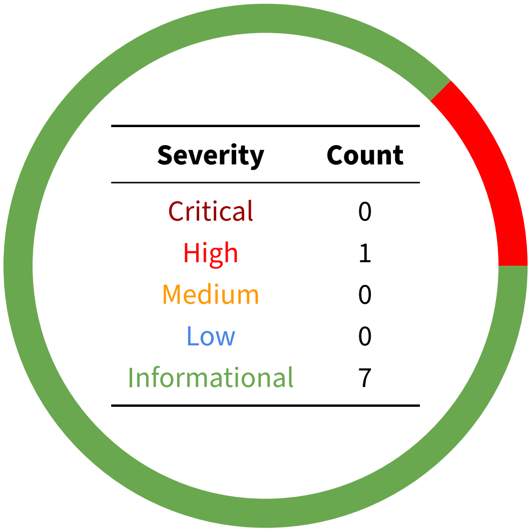
| Name | Description |
|---|---|
| endpoint | Underlying message bridge for LayerZero |
| bridge | Cross-chain asset transfers building on top of `endpoint` |

# 03 | Findings

Overall, we report 8 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.

The below chart displays the findings by severity.



| Severity | Count |
|---|---|
| Critical | 0 |
| High | 1 |
| Medium | 0 |
| Low | 0 |
| Informational | 7 |

# 04 | Vulnerabilities

Here we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have **immediate** security implications, and we recommend remediation as soon as possible.

| ID | Severity | Status | Description |
|---|---|---|---|
| OS-LZR-ADV-00 | High | Resolved | Sending an message from an incorrect UA to the bridge allows an attacker to freeze the bridge |

## OS-LZR-ADV-00 [high] [resolved] | Freeze Bridge with Invalid Sender

### Description

Only the bridge UA is intended to send messages to the bridge contract. However, this behavior is not enforced at the relayer level.

As shown below, any UA can send messages to any other endpoint.

*sources/endpoint.move*

```
public fun send<UA>(dst_chain_id: u64, dst_address: vector<u8>,
    ↪  payload: vector<u8>,fee: Coin<AptosCoin>, _cap:
    ↪  &UaCapability<UA>): (u64, Coin<AptosCoin>) acquires [..] {
        [..]
        let ua_addr = type_address<UA>();
        [..]
        let packet = packet::new_packet(
            global_store.local_chain_id, // src_chain_id
            bcs::to_bytes(&ua_addr),    // src_address
            dst_chain_id,
            dst_address,
            [..]
        );
    }
```

When a different UA sends a message to the bridge endpoint, the message is sent directly to the bridge's packet queue, causing it to be returned when the bridge processes the next packet in `receive`.

As expected, the packet sender is checked afterward, causing the `lz_receive` function to abort in the `assert_trusted_packet` function.

*sources/endpoint.move*

```
public entry fun lz_receive<CoinType>(src_chain_id: u64, receiver:
    ↪  address) acquires CoinStore, GlobalStore {
    [..]
    // receive packet
    let packet = endpoint::receive<BridgeUA>(src_chain_id,
    ↪  &global_store.lz_cap);
    assert_trusted_packet(global_store, &packet, src_chain_id);
    [..]
}
```

```
fun assert_trusted_packet(global_store: &GlobalStore, packet: &Packet,
    ↪  src_chain_id: u64) {
    assert!(table::contains(&global_store.trusted_remotes,
    ↪  src_chain_id),
    ↪  error::not_found(EBRIDGE_REMOTE_BRIDGE_NOT_FOUND));
    let src_bridge_addr = table::borrow(&global_store.trusted_remotes,
    ↪  src_chain_id);
    assert!(packet::src_address(packet) == *src_bridge_addr,
    ↪  error::permission_denied(EBRIDGE_INVALID_REMOTE_BRIDGE));
    }
```

Because the packet is not cleared from the packet queue, it is impossible to retrieve the remaining packets out of the queue, so the bridge will remain frozen.

### Remediation

At its core, the bridge receive function is susceptible to denial of service attacks because any aborts will cause the transaction to revert, leaving the unprocessable packet behind.

To mitigate this particular attack scenario, we discussed it with LayerZero and recommended redesigning the relayer to better isolate message channels. By ensuring message channels are scoped on both `chain_id` and UA address, applications building on top of LayerZero are able to better reason about who they should be processing messages from.

At the same time, we recommend application developers building on top of LayerZero's Aptos endpoints should be aware of this feature, and pay extra attention to the `lz_receive` functionality to avoid potential denial of service scenarios.

### Patch

This issue was mitigated with isolated message channels in commit `efa7cf8`.

*sources/channel.move*

```
struct Channels<phantom UA> has key {
    states: Table<Remote, Channel>
}
```

# 05 | General Findings

Here we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they do represent antipatterns and could introduce a vulnerability in the future.

| ID | Description |
|---|---|
| OS-LZR-SUG-00 | `price_ratio` is unchecked in the configuration function |
| OS-LZR-SUG-01 | Error handling on `table::borrow` is done inconsistently |
| OS-LZR-SUG-02 | Sanity check the submitted oracle hash length |
| OS-LZR-SUG-03 | Inconsistent bridge address size checks |

## OS-LZR-SUG-00 | Inconsistent Config Parameter Checking

### Description

The `layerzero::executor::config` function is checking `airdrop_amt_cap` twice, but not checking `price_ratio`.

*sources/executor.move*

```
public entry fun config(account: &signer, chain_id: u64,
    ↪  airdrop_amt_cap: u64, price_ratio: u64, gas_price: u64)
    ↪  acquires GlobalStore {
    [..]
    assert!(
        airdrop_amt_cap > 0 && airdrop_amt_cap > 0 && gas_price > 0,
        error::invalid_argument(EEXECUTOR_INVALD_CHAINID_U16)
    );
    [..]
}
```

### Remediation

Also add check for `price_ratio` and remove extra check for `airdrop_amt_cap`.

### Patch

These checks were removed in commit `6a6e829`.

# OS-LZR-SUG-01 | Inconsistent Table Error Handling

### Description

In some functions, `table::borrow` and `table::`add are used without error handling.
`table::borrow` will throw an error, if no value exists for the given key.
`table::`add will fail, if there exists a value for the given key already.

*sources/app/counter.move*

```
public entry fun send_to_remote(account: &signer, chain_id: u64, fee:
     ↪  u64) ... {
    [..]
    let config = borrow_global<Config>(signer_addr);
    let dst_address = table::borrow(&config.trusted_remote, chain_id);
    [..]
}

public entry fun set_remote(account: &signer, chain_id: u64, address:
     ↪  vector<u8>) .. {
    [..]
    let config = borrow_global_mut<Config>(signer_addr);
    table::add(&mut config.trusted_remote, chain_id, address);
}
```

### Remediation

Always check if the value exists with `table::contains` before borrowing. In case of table::add, if updating is allowed use `table::upsert`, otherwise use |table::contains| to handle error.

## OS-LZR-SUG-02 | Sanity Check Oracle Hash Size

### Description

Hashes for the packet sizes are usually 32 bytes, so it's worth asserting packet size while submitting the packet.

### Remediation

Assert the hash size in `oracle_submit` function.

*sources/app/oracle.move*

```
public fun oracle_submit<ORACLE>(hash: vector<u8>, confirmations: u64,
    ↪  _cap: &OracleCapability<ORACLE>) acquires OracleStore,
    ↪  GlobalStore {
+     assert!(vector::length<u8>(&hash) == 32, EORACLE_INVALID_HASH);
      let addr = type_address<ORACLE>();
    }
```

### Patch

This check was added in commit `c89ade4`.

*sources/msglib/v1/uln_receive.move*

```
public entry fun oracle_propose(account: &signer, hash: vector<u8>,
    ↪  confirmations: u64) acquires ProposalStore, EventStore {
    assert_length(&hash, 32);
    [..]
}
```

## OS-LZR-SUG-03 | Missing Address Size Checks

### Description

Per the encode payload structure for bridges, sizes should be limited to 32 bytes. However, there is no check to ensure addresses are this size.

*sources/app/bridge.move*

```
// encode payload: packet type(1) + remote token(32) + receiver(32) +
    ↪  amount(8) + unwarp flag(1)
fun encode_send_payload(dst_coin_addr: vector<u8>, dst_receiver:
    ↪  vector<u8>, amount_sd: u64, unwrap: bool): vector<u8> {
    let payload = vector::empty<u8>();
    serde::serialize_u8(&mut payload, PSEND);
    serde::serialize_vector(&mut payload, dst_coin_addr);
    serde::serialize_vector(&mut payload, dst_receiver);
    [..]
    payload
}
```

### Remediation

It is recommended to assert address sizes before encoding.

*sources/app/bridge.move*

```
    assert_bytes32(&dst_coin_addr);
    assert_bytes32(&dst_receiver);
```

# 06 | Formal Verification

Here we present recommendations and example specifications for formal verification of contracts.

| ID | Description |
|---|---|
| OS-LZR-VER-00 | Specifications for bridge address sizing |
| OS-LZR-VER-01 | Specify internal properties of `PacketQueue`, `RemoteCoin` and `Config` |
| OS-LZR-VER-02 | Function invariant specifications. |

## OS-LZR-VER-00 | Bridge Specifications

1. As the address sizes for the bridge is fixed i.e, 32, enforce size checks for addresses by adding specifications.

*sources/app/bridge.move* RUST

```
    spec encode_send_payload {
        aborts_if len<u8>(dst_coin_addr) != 32;
        aborts_if len<u8>(dst_receiver) != 32;
    }

    spec set_remote_bridge {
        aborts_if remote_bridge_addr != 32;
    }
```

2. Explicate when key functions can abort. For example, Queries for checking the existence of a resource should never abort.

*sources/app/bridge.move* RUST

```
    spec has_coin_registered {
        aborts_if false;
    }
```

## OS-LZR-VER-01 | Data Invariant Specifications

1. In packet_queue next_get_nonce should never be greater than next_set_nonce. This relatively nontrivial condition can be easily enforced via a data invariant.

   *sources/packet.move* RUST

   ```
   spec PacketQueue {
       invariant next_set_nonce >= next_get_nonce;
   }
   ```

2. It's worthwhile ensuring sane bounds on the certain critical data fields in `Config`

   *sources/executor.move* RUST

   ```
   spec Config {
       invariant airdrop_amt_cap > 0;
       invariant gas_price > 0;
       invariant price_ratio > 0;
   }
   ```

3. RemoteCoin holds a remote coin bridge address which is a 32byte address. Use a data invariant to ensure the address is valid.

   *sources/executor.move* RUST

   ```
   spec RemoteCoin {
       invariant len<u8>(address) == 32;
   }
   ```

## OS-LZR-VER-02 | Miscellaneous Function Specifications

1. Explicate error conditions to ensure that abort of key functions is well-defined.

*sources/bridge.move* RUST

```
      spec set_treasury_fee {
          aborts_if !exists<FeeStore>(@layerzero);
      }
```

2. A decimal count greater than 22 will cause the pow function to overflow. Explicating such error conditions makes it easier to reason about such code.

*sources/endpoint.move* RUST

```
spec register_coin {
    pragma aborts_if_is_partial = true;
    aborts_if has_coin_registered<CoinType>();
    aborts_if address_of(account) != @bridge;
    aborts_if decimals < 6;
    aborts_if exists<aptos_framework::coin::CoinInfo<CoinType>>
    (address_of(account));
}
```

3. Verify that the next_set_nonce of packet_queue is always greater than next_get_nonce at the end of `insert_next` function execution.

*sources/oracle.move* RUST

```
    spec insert_next {
        ensures packet_queue.next_set_nonce >
     ↪  packet_queue.next_get_nonce;
    }
```