

# LayerZero Aptos V2

## Security Assessment

January 29th, 2025 — Prepared by OtterSec

---

Bartłomiej Wierzbiński

[dark@osec.io](mailto:dark@osec.io)

---

Robert Chen

[r@osec.io](mailto:r@osec.io)

---

# Table of Contents

<b>Executive Summary</b>	<b>3</b>
Overview	3
Key Findings	3
<b>Scope</b>	<b>4</b>
<b>Findings</b>	<b>5</b>
<b>Vulnerabilities</b>	<b>6</b>
OS-LZV-ADV-00   Missing Message Verification	9
OS-LZV-ADV-01   Improper Blocklist Enforcement	10
OS-LZV-ADV-02   Missing Peer Verification	11
OS-LZV-ADV-03   Sender Address Spoofing via Compose Message	12
OS-LZV-ADV-04   Improper DVN Re-initialization Check	13
OS-LZV-ADV-05   Incorrect Parameter Order in Delegate Assertion	14
OS-LZV-ADV-06   Quorum Admin change requires Admin	15
OS-LZV-ADV-07   Rate Limit Issues	16
OS-LZV-ADV-08   Faulty Size Calculations	17
OS-LZV-ADV-09   Overflow Due to Lack of Upscaling	19
OS-LZV-ADV-10   Ability to Set Quorum Value to Zero	20
OS-LZV-ADV-11   Inconsistency in Parameter Order and Utilization	21
OS-LZV-ADV-12   Invalid Size Validation Logic	23
OS-LZV-ADV-13   Absence of Quorum Validation	25
OS-LZV-ADV-14   Improper Ownership management	27
OS-LZV-ADV-15   DVN Duplication and Zero Confirmation Checks	28

OS-LZV-ADV-16	ZRO Configuration Inconsistency	29
OS-LZV-ADV-17	Incorrect Validation Logic	30
OS-LZV-ADV-18	Ambiguity in Function Naming	31
OS-LZV-ADV-19	Version-Specific Fee Calculation	32
<b>General Findings</b>		<b>33</b>
OS-LZV-SUG-00	Functional Incongruity	35
OS-LZV-SUG-01	Counter Oapp Deviation From EVM Implementation	36
OS-LZV-SUG-02	Unverified Cross-Chain Messaging	38
OS-LZV-SUG-03	Burning of Nilified Packets	39
OS-LZV-SUG-04	Inconsistencies in DVN and Executor Modules	40
OS-LZV-SUG-05	Missing Validation Logic	41
OS-LZV-SUG-06	Code Maturity	43
OS-LZV-SUG-07	Code Clarity	44
OS-LZV-SUG-08	Removal of Duplicate Events	45
OS-LZV-SUG-09	Code Refactoring	46
OS-LZV-SUG-10	Error Handling	47
OS-LZV-SUG-11	Separating Production and Test-specific Functionality	48
 <b>Appendices</b>		
<b>Vulnerability Rating Scale</b>		<b>49</b>
<b>Procedure</b>		<b>50</b>

# 01 — Executive Summary

---

## Overview

LayerZero Labs engaged OtterSec to assess the `lz-aptos-v2` program. This assessment was conducted between August 26th, 2024 and January 29th, 2025. For more information on our auditing methodology, refer to [Appendix B](#).

## Key Findings

We produced 32 findings throughout this audit engagement.

In particular, we identified several vulnerabilities related to the lack of proper validation, including the failure to verify the sender in the receive functionality, which allows untrusted sources to send unauthorized messages ([OS-LZV-ADV-02](#)), and the possibility of setting the quorum value to zero ([OS-LZV-ADV-10](#)). Additionally, in the native drop, the account is moved twice, which will result in an error ([OS-LZV-ADV-14](#)). There is also a critical flaw concerning insufficient message verification, where OFT cores receive function fails to call the endpoints clear function to validate incoming data ([OS-LZV-ADV-00](#)).

Furthermore, there are multiple instances where an incorrect parameter is utilized or certain parameters are omitted, resulting in inconsistencies in functionality and event logging ([OS-LZV-ADV-11](#)), and the assert delegate function is called with the parameters in the wrong order, passing the OApp address first instead of the delegate address ([OS-LZV-ADV-05](#)). We also highlighted faulty size calculations in several functions ([OS-LZV-ADV-08](#)), and a possible overflow scenario due to exceeding the data type's maximum permissible value ([OS-LZV-ADV-09](#)).

We also recommended modifying the codebase to mitigate potential security issues and improve functionality ([OS-LZV-SUG-09](#)), and made suggestions regarding inconsistencies in the codebase while ensuring adherence to best development practices ([OS-LZV-SUG-06](#)). We further advised incorporating additional checks for improved robustness and security ([OS-LZV-SUG-05](#)) and streamlining the code to reduce complexity, eliminate redundancy, and enhance readability ([OS-LZV-SUG-08](#)).

# 02 — Scope

The source code was delivered to us in a Git repository at <https://github.com/LayerZero-Labs/monorepo>. This audit was performed against commit [c0e64d0](#).

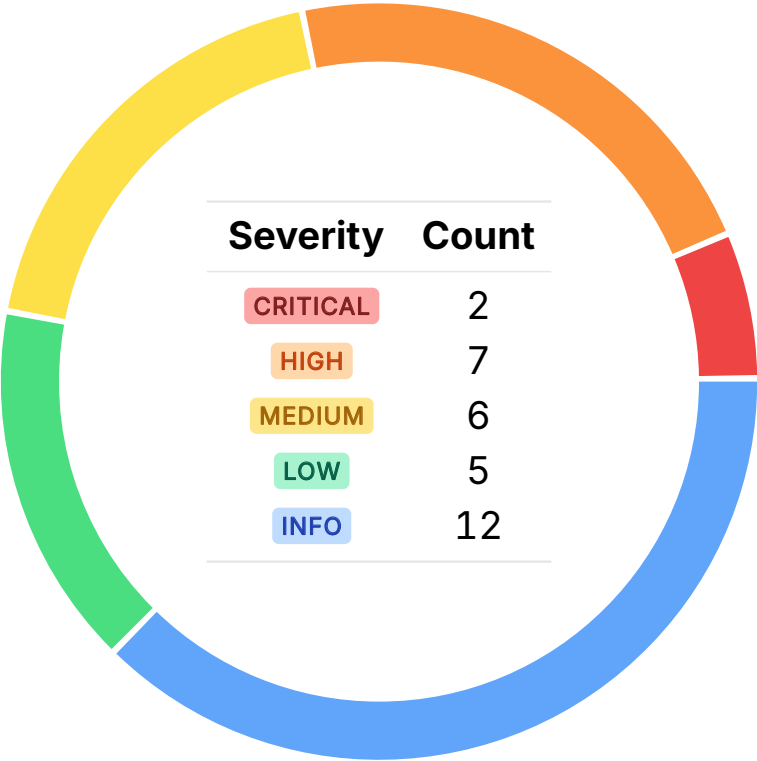
A brief description of the programs is as follows:

Name	Description
endpoint	Core protocol managing trustless security and flow.
uln 302	Manages trust flow between workers like the Executor and DVN.
routing-layer	Handles communication with message libraries, price feed modules, and fee library modules, replacing the need for dynamic dispatch.
unified-price-feed	Provides a platform for publishing and retrieving prices across the ecosystem, ensuring accurate conversion of fees into the local native currency.
unified-fee-fibrary	Computes fees for workers including the executor and DVNs.
treasury	Configures and manages treasury fees.
worker-registry	It manages configuration of DVN and Executor settings.
oft	A cross-chain token built on top of the LayerZero ecosystem.

# 03 — Findings

Overall, we reported 32 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 do not have an immediate impact but will aid in mitigating future vulnerabilities.



## 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.

Rating criteria can be found in [Appendix A](#).

ID	Severity	Status	Description
OS-LZV-ADV-00	CRITICAL	RESOLVED ✓	<code>oft_core::receive</code> lacks proper message verification, as it does not call <code>endpoint::clear</code> to validate incoming data.
OS-LZV-ADV-01	CRITICAL	RESOLVED ✓	<code>debit_fungible_asset</code> improperly utilizes <code>redirect_to_admin_if_blocklisted</code> , designed for receiving funds, instead of <code>assert_not_blocklisted</code> , allowing block-listed addresses to send assets.
OS-LZV-ADV-02	HIGH	RESOLVED ✓	<code>lz_receive</code> lacks proper sender verification, allowing untrusted sources to send unauthorized messages.
OS-LZV-ADV-03	HIGH	RESOLVED ✓	<code>send_coin</code> allows for potential sender address spoofing, as it accepts the sender as an external parameter, enabling unauthorized manipulation of the <code>compose_message</code> .
OS-LZV-ADV-04	HIGH	RESOLVED ✓	<code>initialize</code> in the DVN module incorrectly checks if the <code>vid_store</code> is not zero, disallowing the initialization process.
OS-LZV-ADV-05	HIGH	RESOLVED ✓	<code>assert_delegate</code> is called with the parameters in the wrong order, passing the <code>OApp</code> address first instead of the delegate address.

OS-LZV-ADV-06	HIGH	RESOLVED ✓	<code>quorum_change_admin</code> requires the admin that needs to be added or removed to be the caller, rendering it impossible to remove an admin without their direct involvement, which defeats the purpose of quorum-based control.
OS-LZV-ADV-07	HIGH	RESOLVED ✓	<code>set_rate_limit_at_timestamp</code> may create a temporary denial of service if the new rate limit is set lower than the current <code>in_flight_on_last_update</code> , resulting in blocked capacity in rate-limited features. Also, there is no functionality to unset the rate limit.
OS-LZV-ADV-08	HIGH	RESOLVED ✓	In <code>executor_options</code> , the size calculations are faulty in several functions.
OS-LZV-ADV-09	MEDIUM	RESOLVED ✓	Intermediate calculations in <code>oft_impl_config</code> may exceed the data type's maximum value, resulting in an overflow.
OS-LZV-ADV-10	MEDIUM	RESOLVED ✓	<code>set_quorum</code> allows setting the <code>quorum</code> value to zero.
OS-LZV-ADV-11	MEDIUM	RESOLVED ✓	There are multiple cases where incorrect parameters are utilized or certain parameters are not included, resulting in inconsistencies in functionality and event logging.
OS-LZV-ADV-12	MEDIUM	RESOLVED ✓	<code>extract_legacy_options</code> contains a flawed size check for <code>option_type == 2</code> , utilizing an OR condition instead of an AND, which results in the validation to always pass, potentially allowing invalid option sizes.

OS-LZV-ADV-13	MEDIUM	RESOLVED ✓	<code>set_multisig_signers</code> does not validate the <code>quorum</code> , which could allow for setting the signer count below <code>quorum</code> threshold.
OS-LZV-ADV-14	MEDIUM	RESOLVED ✓	In <code>executor::native_drop</code> , <code>account</code> is moved twice, which will result in an error.
OS-LZV-ADV-15	LOW	RESOLVED ✓	The ULN configuration does not prevent marking a DVN as both <code>required</code> and <code>optional</code> , and it allows setting zero confirmations.
OS-LZV-ADV-16	LOW	RESOLVED ✓	The <code>ZRO</code> configuration was moved to <code>UniversalStore</code> , but admin controls in <code>endpoint_v2::admin</code> still point to the outdated <code>EndpointStore</code> . As a result the <code>set_zro_enabled</code> flag and <code>zro_enabled</code> view function reference unutilized data.
OS-LZV-ADV-17	LOW	RESOLVED ✓	<code>get_executor_config</code> incorrectly checks for <code>option::is_some(&amp;default_config)</code> .
OS-LZV-ADV-18	LOW	RESOLVED ✓	<code>is_blocklisted</code> and <code>is_block_listed</code> have similar names. Since <code>is_blocklisted</code> blocks the coin, it may result in potential confusion and accidental changes to the <code>blocklist</code> status.
OS-LZV-ADV-19	LOW	RESOLVED ✓	<code>quote_send</code> fails to differentiate between v1 and v2 message formats, potentially resulting in inaccurate network fee calculations and allowing quotes for unsupported message types.

## Missing Message Verification CRITICAL

OS-LZV-ADV-00

### Description

`oft_core::receive` does not verify the authenticity or integrity of the incoming message before processing it. The function accepts incoming messages and processes them without checking whether they are legitimate, authentic, or have already been processed (replay protection). Without verification, a malicious actor may replay a previous message, and tamper with the message's payload.

```
>_ oft/sources/internal/oft_core.move
```

RUST

```
public(friend) inline fun receive(  
    src_eid: u32,  
    nonce: u64,  
    guid: Bytes32,  
    message: vector<u8>,  
    lz_receive_fa: Option<FungibleAsset>,  
    credit: |address, u64, Option<FungibleAsset>| u64,  
) {  
    let to_address = bytes32::to_address(oft_msg_codec::send_to(&message));  
    let message_amount_ld = oft_store::to_ld(oft_msg_codec::amount_sd(&message));  
    let amount_received_ld = credit(  
        to_address,  
        message_amount_ld,  
        lz_receive_fa  
    );  
    [...]  
}
```

### Remediation

Include a call to `endpoint::clear` to ensure that the message parameters are properly verified.

### Patch

Resolved in [60048b9](#) by adding the call to `endpoint::clear`.

## Improper Blocklist Enforcement CRITICAL

OS-LZV-ADV-01

### Description

In the current implementation of `oft_fa::debit_fungible_asset`, the function calls `redirect_to_admin_if_blocklisted`, which checks the blocklist status of the sender. While this behavior is suitable for receive operations, where funds may be redirected to the admin if the recipient is blocklisted, it is inappropriate for debit operations initiated by the sender. In such cases, the transaction should be blocked entirely if the sender is blocklisted.

```
>_ oft/sources/oft_implementation/oft_fa.move
```

RUST

```
public(friend) fun debit_fungible_asset(  
  sender: address,  
  fa: &mut FungibleAsset,  
  min_amount_ld: u64,  
  dst_eid: u32,  
) : (u64, u64) acquires OftImpl {  
  redirect_to_admin_if_blocklisted(sender, min_amount_ld);  
  try_consume_rate_limit_capacity(dst_eid, min_amount_ld);  
  [...]  
}
```

### Remediation

Utilize `assert_not_blocklisted` to ensure that blocklisted addresses are restricted from performing any debit action.

### Patch

Resolved in [8ac47a7](#).

## Missing Peer Verification HIGH

OS-LZV-ADV-02

### Description

Currently, in `oapp_receive::lz_receive`, there is a lack of authentication when receiving cross-chain messages. Specifically, the function currently does not verify whether the sender of the message is a trusted peer. This opens up the system to potential attacks where an untrusted entity may send unauthorized messages, which will be processed as legitimate by the receiving contract.

```
>_ oapps/oft/sources/internal_oapp/oapp_receive.move
```

RUST

```
public entry fun lz_receive(
    [...]
) {
    let sender = to_bytes32(sender);
    let guid = to_bytes32(guid);
    endpoint::clear(&oapp_store::call_ref(), src_eid, sender, nonce, wrap_guid(guid), message);
    lz_receive_impl(
        src_eid,
        sender,
        nonce,
        guid,
        message,
        extra_data,
        option::none(),
    )
}
```

### Remediation

Include a peer verification check to ensure that the sender of the message is a trusted peer for that source endpoint.

### Patch

Resolved in [2a4f7cf](#).

## Sender Address Spoofing via Compose Message

HIGH

OS-LZV-ADV-03

### Description

In `oft_coin::send_coin` (shown below) and `oft_fa::send_fa`, the sender's address is passed as an argument and then utilized to build the transaction message ( `compose_message` ). This message is critical because it contains the details of the cross-environment transfer, such as the recipient address ( `to` ), the amount ( `send_value` ), and other metadata. However, since the sender's address is provided as an input parameter, it implies that it may be spoofed by the caller of the function. Consequently, a malicious actor could impersonate another user, manipulating the `compose_message` to make it appear that a different, legitimate sender is initiating the transfer.

```
>_ oapps/oft/sources/internal_oft/oft.move
```

RUST

```
public fun send_coin(  
  sender: address,  
  dst_eid: u32,  
  to: Bytes32,  
  send_value: &mut Coin<Oft>,  
  min_amount_ld: u64,  
  extra_options: vector<u8>,  
  compose_message: vector<u8>,  
  oft_cmd: vector<u8>,  
  native_fee: &mut FungibleAsset,  
  zro_fee: &mut Option<FungibleAsset>,  
) : (MessagingReceipt, OftReceipt) {  
  [...]  
}
```

### Remediation

Utilize `get_dynamic_call_ref_caller` instead of taking the sender's address as a parameter.

### Patch

Resolved in [10deb8c](#) and [ae4791d](#).

## Improper DVN Re-initialization Check HIGH

OS-LZV-ADV-04

### Description

`initialize` in `dvn` utilizes an incorrect assertion. Instead of utilizing `==`, the check is utilizing `!=`.

```
>_ dvn/sources/dvn.move
```

RUST

```
public entry fun initialize(  
  [...]   
) acquires DvnStore {  
  [...]   
  // Set the VID (endpoint v1-v2 compatible eid e.g. 30101 -> 101)  
  // This is also used as a flag to indicate that the DVN has been initialized  
  assert!(vid != 0, EDVN_INVALID_VID);  
  let vid_store = &mut store_mut().vid;  
  assert!(*vid_store != 0, EALREADY_INITIALIZED);  
  *vid_store = vid;  
}
```

`vid_store` is set to zero upon creation. Thus, the subsequent assertion on `vid_store`, which requires it to be non-zero, will never pass, thereby inhibiting the initialization. During the first call to `initialize`, `vid_store` will always be zero, and the assertion should be adjusted accordingly to account for this scenario.

### Remediation

Update the initialization check to `vid_store == 0` to ensure the function is only called once (initialization phase).

### Patch

Resolved in [0029328](#).

## Incorrect Parameter Order in Delegate Assertion HIGH

OS-LZV-ADV-05

### Description

There is a mismatch in the order of parameters passed to `assert_delegate` within `oapp_delegate`. Specifically, `assert_delegate` expects its parameters in the order of `(delegate, oapp)`, but in the existing code, it is called with the parameters in the reverse order, `(oapp, delegate)`. The code snippet below illustrates the incorrect call in `nilify`. Since the parameters are reversed, this check does not function as intended, because the wrong values are compared, causing it to always fail.

```
>_ oapp_core/sources/oapp_delegate.move
```

RUST

```
/// Nilify an OApp message
public entry fun nilify(
    [...]
) acquires DelegateStore {
    assert_delegate(oapp, address_of(move account));
    endpoint::nilify(&call_ref(oapp), src_eid, to_bytes32(sender), nonce,
        ↪ to_bytes32(payload_hash))
}

/// Assert that an address is the delegate for an OApp
inline fun assert_delegate(delegate: address, oapp: address) acquires DelegateStore {
    assert!(delegate(oapp) == delegate, EUNAUTHORIZED);
}
```

### Remediation

Ensure the parameters are passed in the correct order.

### Patch

Resolved in [1ad3421](#) by removing `oapp_delegate`.

## Quorum Admin change requires Admin HIGH

OS-LZV-ADV-06

### Description

The vulnerability arises from how `dvn::quorum_change_admin` handles the addition or removal of an admin. Specifically, the admin to be added or removed must invoke `quorum_change_admin`. This contradicts the idea of quorum-based decision-making because the admin may block their removal by simply refusing to initiate the transaction.

```
>_ contracts/workers/dvn/sources/dvn.move
```

RUST

```
public entry fun quorum_change_admin(  
  account: &signer,  
  active: bool,  
  expiration: u64,  
  signatures: vector<u8>,  
) acquires DvnStore {  
  let admin = address_of(move account);  
  let hash = create_quorum_change_admin_hash(admin, active, get_vid(), expiration);  
  assert_all_and_add_to_history(call_ref(), &signatures, expiration, hash);  
  worker_config::set_worker_admin(call_ref(), admin, active);  
}
```

`quorum_change_admin` is intended to rely on the signatures of multiple DVN signers (quorum) to authorize admin changes, not the signer calling the function. This ensures that a group decision is made, rather than allowing individual signers to act unilaterally.

### Remediation

Accept an `admin` argument (the address of the admin to be added or removed) and utilize the quorum of signatures to authorize the addition or removal of the specified admin, without relying on the admin themselves to initiate the transaction.

### Patch

Resolved in [d7120ce](#).

## Rate Limit Issues HIGH

OS-LZV-ADV-07

### Description

There is a potential denial-of-service risk in

`oft_impl_config::set_rate_limit_at_timestamp`, particularly when the new rate limit that is set for an `EID` (Endpoint ID) is lower than the current `in_flight_on_last_update` value. Since the `in_flight_on_last_update` is preserved (not reduced) when the limit is set, the system may erroneously assume it has already used more capacity than available, which could cause it to exceed the newly set rate limit.

```
>_ oft/sources/internal_oft/oft_impl_config.move
```

RUST

```
/// Set or update the rate limit for a given EID at a specified timestamp
public(friend) fun set_rate_limit_at_timestamp(
    [...]
) acquires Config {
    assert!(window_seconds > 0, EINVAL_WINDOW);
    if (has_rate_limit(dst_eid)) {
        checkpoint_rate_limit_in_flight(dst_eid, timestamp);
        let rate_limit_store = table::borrow_mut(&mut store_mut().rate_limit_by_eid, dst_eid);
        rate_limit_store.limit = limit;
        rate_limit_store.window_seconds = window_seconds;
        emit(RateLimitUpdated { dst_eid, limit, window_seconds });
    } [...]
}
```

Furthermore, in the current implementation, once a rate limit is set, there is no functionality to remove or reset the limit. `unset_rate_limit` is not utilized anywhere. If the rate limit is set to zero, it will essentially freeze the system as no decay will occur. Additionally, if the rate limit is set to an extremely high value, it may result in a denial-of-service scenario, especially if this value is multiplied without proper upscaling.

### Remediation

Set the `in_flight` value to `limit` if `limit < in_flight_on_last_update`, or check for this edge case in `rate_limit_capacity`. Also, allow unsetting the rate limit via `unset_rate_limit`.

### Patch

Resolved in [f64de68](#).

## Faulty Size Calculations HIGH

OS-LZV-ADV-08

### Description

Within `executor_options`, there are multiple instances where the size calculations are incorrect. In `append_lz_receive_option`, `size` should return 17 or 33, not 16 or 32. `option_type` and the gas amount take 17 bytes total, or 33 if an additional 16-byte value is appended.

```
>_ worker_peripherals/fee_libs/executor_fee_lib_0/sources/types/executor_option.move RUST

fun append_lz_receive_option(output: &mut vector<u8>, lz_receive_option: &LzReceiveOption) {
    let size = if (lz_receive_option.value == 0) { 16 } else { 32 };
    serde::append_u16(output, size);
    [...]
}
```

Similarly, in `append_lz_compose_option`, `size` should return either 19 or 35, instead of 18 or 34. `option_type` takes 1 byte, `index` takes 2 bytes, and gas amount is a 16-byte value (19 bytes in total). If a 16-byte value is appended, the total becomes 19+16 = 35. Additionally, `append_ordered_execution_option` should utilize a `size` of 1 and not 0 as `option_type` takes 1 byte.

```
>_ worker_peripherals/fee_libs/executor_fee_lib_0/sources/types/executor_option.move RUST

fun append_ordered_execution_option(output: &mut vector<u8>) {
    serde::append_u16(output, 0); // size = 0
    serde::append_u8(output, OPTION_TYPE_ORDERED_EXECUTION);
}
```

In `append_native_drop_option`, there is a mismatch between the actual size of the serialized data and the size that is appended to the buffer. Currently, 48 is appended, which will result in problems when deserializing the data because the deserializer will expect 48 bytes but will encounter 49 bytes of actual data. It should append a size of 49 instead of 48, as the summation comes out to be 49 bytes (1 byte ( `option` ) + 16 bytes ( `amount` ) + 32 bytes ( `receiver` )).

```
>_ worker_peripherals/fee_libs/executor_fee_lib_0/sources/types/executor_option.move RUST

fun append_native_drop_option(output: &mut vector<u8>, native_drop_option: &NativeDropOption) {
    serde::append_u16(output, 48);
    serde::append_u8(output, OPTION_TYPE_NATIVE_DROP);
    [...]
}
```

Furthermore, in `deserialize_executor_options` the `option_size` field does not account for the presence of the `option_type` when determining the size of the payload. As executor options are constructed in a `[worker_id][option_size][option_type][payload]` pattern, `deserialize_executor_options` should subtract the length of the `option_type` field before extracting the `payload`.

## Remediation

Incorporate the above refactors to ensure that all size calculations are performed correctly.

## Patch

Resolved in [fa7f545](#) and [c971bd3](#).

## Overflow Due to Lack of Upscaling MEDIUM

OS-LZV-ADV-09

### Description

There is a risk of overflow in mathematical operations within functions such as `to_ld`, `debit_view_with_possible_fee`, `in_flight_at_timestamp`, and `try_consume_rate_limit_capacity_at_timestamp` in `oft_impl_config`. These issues arise from improper scaling during arithmetic operations, particularly when handling large numbers, where multiplying large variables may exceed the maximum limit of the data type.

### Remediation

Provide headroom for large values during multiplication by properly upscaling to prevent overflow issues.

### Patch

Resolved in [f4bf07f](#).

## Ability to Set Quorum Value to Zero MEDIUM

OS-LZV-ADV-10

### Description

Within `multisig_store::set_quorum`, it is possible to set the `quorum` value to zero. As a result, operations that require signatures may be called freely by passing an empty signatures vector. This further implies that the guarded functions may be called without proper verification.

```
>_ worker_common/sources/internal/multisig_store.move
```

RUST

```
public(friend) fun set_quorum(worker: address, quorum: u64) acquires SigningStore {  
    let worker_store = signing_store_mut(worker);  
    let signer_count = vector::length(&worker_store.signers);  
    assert!(quorum <= signer_count, ESIGNERS_LESS_THAN_QUORUM);  
    worker_store.quorum = quorum;  
}
```

### Remediation

Disallow setting the quorum value to zero in `set_quorum`.

### Patch

Resolved in [d7120ce](#).

## Inconsistency in Parameter Order and Utilization

MEDIUM

OS-LZV-ADV-11

### Description

In `send::pay_executor_and_assert_size`, while emitting the `emit_executor_fee_paid` event, the `sender` is utilized as the first parameter instead of the `executor_address`. As a result, the event inaccurately reflects who received the fee, which may lead to misunderstandings about which party actually performed the service for which the fee was paid.

```
>_ uln_302/sources/internal/sending.move
```

RUST

```
public(friend) inline fun pay_executor_and_assert_size(
  sender: address,
  native_token: &mut FungibleAsset,
  dst_eid: u32,
  message_length: u64,
  executor_options: vector<u8>,
  get_executor_fee: |address, vector<u8>| (u64, address),
): u64 {
  [...]
  // Emit a record of the fee paid to the executor
  emit_executor_fee_paid(sender, deposit_address, fee);
  fee
}
```

Additionally, in `oft_core::send` in `oft::send_fa`, `peer` is invoked with only one parameter (`dst_eid`), whereas it expects two parameters. Furthermore, `oft_adapter::debit` (shown below) currently transfers `amount_ld` (the original amount passed as an argument) to the escrow address. This is incorrect because the actual amount to be transferred should have been calculated by `debit_view`, which takes into account possible adjustments such as fees, slippage checks, and dust removal, consequently returning the correct `amount_ld`.

```
>_ oft/sources/oft_adapter.move
```

RUST

```
fun debit(from: &signer, amount_ld: u64, min_amount_ld: u64, dst_eid: u32): (u64, u64)
  => acquiresManagingRefs {
  let (amount_send_ld, amount_received_ld) = debit_view(amount_ld, min_amount_ld, dst_eid);
  // lock the amount in escrow
  let escrow_address = object::address_from_extend_ref(borrow_escrow_extend_ref());
  primary_fungible_store::transfer(from, oft_core::metadata(), escrow_address, amount_ld);
  (amount_send_ld, amount_received_ld)
}
```

## Remediation

Ensure that the `executor_address` is utilized instead of the `sender` for the first parameter of the `emit_executor_fee_paid` event. Also, include all necessary parameters in `peer` and utilize `debit_view` in `oft_adapter::debit` to calculate `amount_ld`.

## Patch

1. Resolved in [8ca9f5a](#).
2. Resolved in [655032c](#).
3. Resolved in [6b7475d](#).

## Invalid Size Validation Logic MEDIUM

OS-LZV-ADV-12

### Description

The issue in `oft::extract_legacy_options` arises from the incorrect utilization of the logical OR (`||`) operator in the condition for checking the size of the options vector when `option_type == 2`. When `total_options_size > 66` is true, even if the size is too large (greater than 98), this part of the condition will pass, and the assertion will succeed, resulting in the function continuing execution even if the size is invalid.

```
>_ oapps/oft/sources/internal_oft/oft.move
```

RUST

```
/// Extracts options in legacy formats
/// @return (executor_options, dvn_options)
public fun extract_legacy_options(option_type: u16, options: &vector<u8>): (vector<u8>,
    → vector<u8>) {
    [...]
    if (option_type == 1) {
        assert!(total_options_size == 34, EINVALID_LEGACY_OPTIONS_TYPE_1);
        let execution_gas = (serde::extract_u256(options, &mut position) as u128);
        append_legacy_option_lz_receive(&mut executor_options, execution_gas);
    } else if (option_type == 2) {
        assert!(total_options_size > 66 || total_options_size <= 98,
            → EINVALID_LEGACY_OPTIONS_TYPE_2);
        let execution_gas = (serde::extract_u256(options, &mut position) as u128);
        [...]
    } else {
        abort EINVALID_OPTION_TYPE
    };
    (executor_options, vector[])
}
```

Similarly, when `total_options_size <= 98` is true, even if the size is too small (less than 66), the second part of the condition will pass, rendering the assertion ineffective in rejecting invalid sizes. As a result, it is impossible for the `EINVALID_LEGACY_OPTIONS_TYPE_2` error to be triggered, allowing incorrectly sized options to pass validation.

### Remediation

Enforce a strict range check where the `total_options_size` must be greater than 66 bytes and less than or equal to 98 bytes by replacing the OR (`||`) operator with an AND (`&&`) operator.

## Patch

Resolved in [cfc1912](#).

## Absence of Quorum Validation MEDIUM

OS-LZV-ADV-13

### Description

`multisig_store::set_multisig_signers` does not include any logic to verify that the number of signers being set meets or exceeds the `quorum` threshold. This implies that after setting a `quorum` value, a worker may set a new list of signers that has fewer signers than the previously defined `quorum`. Thus, if the `quorum` is set to a value (such as 3), but the list of signers is updated to include only 2 signers, this creates a situation where the `quorum` requirement will not be met for any subsequent operations that require signatures.

```
>_ worker_common/sources/internal/multisig_store.move
```

RUST

```
public(friend) fun set_multisig_signers(
  worker: address,
  multisig_signers: vector<vector<u8>>,
) acquires SigningStore {
  assert_no_duplicates(&multisig_signers);
  vector::for_each_ref(&multisig_signers, |signer| {
    assert!(vector::length(signer) == 64, EINVALID_SIGNER_LENGTH);
  });
  signing_store_mut(worker).signers = multisig_signers;
}
```

Consequently, these operations will fail to execute, as the check in `assert_signatures_verified_internal` will not pass, since the `signatures_count` is less than the `quorum` value.

```
>_ worker_common/sources/internal/multisig_store.move
```

RUST

```
public(friend) fun assert_signatures_verified_internal(
  signatures: &vector<vector<u8>>,
  hash: vector<u8>,
  multisig_signers: &vector<vector<u8>>,
  quorum: u64,
) {
  let signatures_count = vector::length(signatures);
  assert!(signatures_count >= quorum, EDVN_LESS_THAN_QUORUM);
  [...]
}
```

### Remediation

Implement a check in `set_multisig_signers` to ensure that the number of signers being set is at least equal to or greater than the quorum requirement.

## Patch

Resolved in [2fd3c4d](#) by adding the quorum check.

## Improper Ownership management MEDIUM

OS-LZV-ADV-14

### Description

In `executor::native_drop`, `account` is moved twice: once when transferring the ownership of `account` to `address_of`, and again when `account` is passed to `primary_fungible_store::withdraw`. Since it was already moved in `assert_admin(address_of(move account))`, attempting to move it again will result in an error.

```
>_ workers/executor/sources/executor.move
```

RUST

```
public entry fun native_drop(  
  [...]   
) {  
  assert_admin(address_of(move account));  
  [...]   
  // Last signer use  
  let fa_total = primary_fungible_store::withdraw(move account, metadata, total_amount);  
  [...]   
}
```

### Remediation

Ensure the `account` in `native_drop` is not moved twice.

### Patch

Resolved in [56e51be](#).

## DVN Duplication and Zero Confirmation Checks LOW

OS-LZV-ADV-15

### Description

Currently, `assert_valid_uln_config` and `assert_valid_default_uln_config` check for duplicates within the `required_dvns` list and the `optional_dvns` list. However, they do not check for duplicates across these lists. A Domain Validation Node (DVN) should not be listed as both required and optional.

Also, the configuration currently allows setting the confirmations value to zero. Confirmations are necessary to ensure that a transaction or message is properly validated by multiple nodes. Allowing zero confirmations defeats the purpose of multi-node validation.

### Remediation

Ensure that `assert_valid_uln_config` and `assert_valid_default_uln_config` cross-check between the `required_dvns` and `optional_dvns` lists to ensure that no DVN appears in both, and restrict setting the confirmation value to zero.

### Patch

Resolved in [b612f0f](#).

## ZRO Configuration Inconsistency LOW

OS-LZV-ADV-16

### Description

In the current implementation, the **ZRO** flag is stored in **UniversalStore**, but **admin::set\_zro\_enabled** is still interacting with the outdated field in **EndpointStore**. This implies that any changes made by this admin function are applied to the wrong storage structure ( **EndpointStore** ) rather than the correct one ( **UniversalStore** ), creating a data consistency issue. The actual **ZRO** configuration is controlled by **UniversalStore**, but the admin tool is updating **EndpointStore**.

```
>_ endpoint_v2_common/sources/universal_config.move
```

RUST

```
struct UniversalStore has key {  
    // The EID for this endpoint  
    eid: u32,  
    // The ZRO metadata if it has been set  
    zro_data: Option<Object<Metadata>>,  
    // Whether the ZRO address is locked. Once locked the zro metadata cannot be changed  
    zro_locked: bool,  
}
```

Therefore, any admin who thinks they are controlling **ZRO** by utilizing **set\_zro\_enabled** is, in fact, modifying data that is no longer utilized. Furthermore, the getter function in **Endpointstore** ( **endpoint::zro\_enabled** ) returns the status of **ZRO**, which is misleading.

```
>_ endpoint_v2/sources/admin.move
```

RUST

```
public entry fun set_zro_enabled(admin: &signer, enabled: bool) {  
    assert_admin(address_of(move admin));  
    store::set_zro_enabled(enabled);  
}
```

### Remediation

Remove the deprecated **EndpointStore** storage logic for **ZRO** from the codebase to prevent accidental utilization.

### Patch

Resolved in [d7120ce](#).

## Incorrect Validation Logic LOW

OS-LZV-ADV-17

### Description

In `configuration::get_executor_config`, `else if` block checks for the same condition as the `else` assert.

```
>_ msglib/libs/uln_302/sources/internal/configuration.move
```

RUST

```
public(friend) fun get_executor_config(sender: address, eid: u32): ExecutorConfig {  
    [...]  
    if (option::is_some(&oapp_config) && option::is_some(&default_config)) {  
        // get correct merged config if oapp and default are both set  
        merge_executor_configs(option::borrow(&default_config), option::borrow(&oapp_config))  
    } else if (option::is_some(&default_config)) {  
        option::extract(&mut default_config)  
    } else {  
        // provide default if oapp does not have its own configuration  
        assert!(option::is_some(&default_config), EEID_NOT_CONFIGURED);  
        option::extract(&mut default_config)  
    }  
}
```

### Remediation

Refactor the `if` conditions to implement a proper hierarchy of validations.

### Patch

Resolved in [31c8e04](#).

## Ambiguity in Function Naming LOW

OS-LZV-ADV-18

### Description

In `oft_coin`, the naming convention of the two functions, `is_blocklisted` and `is_block_listed` may result in confusion and potential mistakes in how the functions are used, especially since both are public functions with similar names but distinct purposes. `is_blocklisted` blocks the coin, so it should be renamed to `set_blocklist` as done in other `OFT` implementations.

```
>_ oft/sources/oft_implementation/oft_coin.move
```

RUST

```
public entry fun is_blocklisted(admin: &signer, wallet: address, block: bool) acquires OftImpl {
  assert_admin(address_of(admin));
  oft_impl_config::set_blocklist(wallet, block);

  // Additionally freeze the Coin store if the wallet is blocked
  if (block) {
    coin::freeze_coin_store(wallet, &store<PlaceholderCoin>().freeze_cap);
  } else {
    coin::unfreeze_coin_store(wallet, &store<PlaceholderCoin>().freeze_cap);
  }
}

#[view]
/// Get the blocklist status of a wallet address
public fun is_block_listed(wallet: address): bool { oft_impl_config::is_blocklisted(wallet) }
```

Specifically, if `is_blocklisted` is accidentally called by the admin instead of `is_block_listed`, it will result in the blocking or unblocking of the coin. On the other hand, `is_block_listed` checks the `blocklist` status, returning true if the wallet is blocklisted and false otherwise. Thus, its name clearly reflects its intent.

### Remediation

Rename `is_blocklisted` to `set_blocklist` to align the function's name with its actual behavior of setting or modifying the `blocklist` status.

### Patch

Resolved in [8ac47a7](#).

## Version - Specific Fee Calculation LOW

OS-LZV-ADV-19

### Description

There is a discrepancy in how quotes are calculated between versions (v1 and v2) and in the handling of unsent messages during quoting. Specifically, in v1, `quote_send` may incorrectly include the `compose` message in the message size, resulting in inflated size and fee calculations. Additionally, while it is possible to quote the `compose` message when utilizing a v1 endpoint, sending it is not allowed, which is inconsistent.

### Remediation

Implement logic in `quote_send` that detects which version of the protocol is utilized. This will help apply the correct message size and quote calculation logic.

### Patch

Resolved in [f64de68](#).

## 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 represent anti-patterns and may result in security issues in the future.

ID	Description
OS-LZV-SUG-00	<code>next_nonce_impl</code> returns the maximum received nonce, which may result in invalid nonce validation in <code>accept_nonce</code> .
OS-LZV-SUG-01	<code>lz_receive_impl</code> in the Move implementation lacks proper validation to ensure that the <code>receive_value</code> matches the value encoded in the message, and also passes different values to the message and executor options.
OS-LZV-SUG-02	<code>endpoint::send_compose</code> allows any <code>OApp</code> to trigger the <code>lz_compose</code> of another <code>OApp</code> without sufficient verification of the sender's legitimacy.
OS-LZV-SUG-03	Nilified packets may still be burned if the nonce passes the lazy inbound nonce.
OS-LZV-SUG-04	There are multiple inconsistencies in the setters and getters for DVN and executor settings.
OS-LZV-SUG-05	There are several instances where proper validation is not performed, resulting in potential security issues.
OS-LZV-SUG-06	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.
OS-LZV-SUG-07	The overall code may be streamlined further to reduce complexity, eliminate redundancy, and enhance readability.
OS-LZV-SUG-08	There are multiple scenarios where duplicate events may be registered.

OS-LZV-SUG-09	Recommendations for modifying the codebase to mitigate potential security issues and improve functionality.
OS-LZV-SUG-10	The error handling may be improved to provide clearer feedback in the event of failures.
OS-LZV-SUG-11	In <code>oft_impl_config</code> dedicated test functions should allow rate limit functions to be called with any timestamp for better flexibility and readability.

## Functional Incongruity

OS-LZV-SUG-00

### Description

The current implementation of the `counter::next_nonce_impl` retrieves the next nonce given the maximum received nonce for a specific sender and source endpoint ID ( `EID` ) from the counter module's data structure ( `counter().max_received_nonce` ).

```
>_ oapps/counter/sources/counter.move
```

RUST

```
public(friend) fun next_nonce_impl(src_eid: u32, sender: Bytes32): u64 acquires Counter {  
    *table::borrow_with_default(&counter().max_received_nonce, ReceivePathway { src_eid, sender  
        ↪ }, &0)  
}
```

However, it does not return the correct nonce, as it should return `max_received_nonce` plus one, since `accept_nonce` verifies whether the incoming nonce is the next in line.

### Remediation

Modify `next_nonce_impl` to return the current maximum nonce plus one.

### Patch

Resolved in [b53d977](#).

## Counter Oapp Deviation From EVM Implementation

OS-LZV-SUG-01

### Description

In the Move implementation in `counter::lz_receive_impl`, when receiving an `ABA_TYPE` message, the contract does not properly validate whether the `receive_value` (an optional fungible asset) corresponds to what was actually sent in the message. This introduces a potential vulnerability where the contract may proceed with an insufficient or incorrect asset transfer. In contrast, the EVM counterpart (`OmniCounterAbstract`) explicitly performs this check, ensuring that the value received matches the value encoded in the message.

```
>_ oapps/counter/sources/counter.move
```

RUST

```
// The Move implementation
public(friend) fun lz_receive_impl(
    [...]
) acquires Counter {
    accept_nonce(src_eid, sender, nonce);
    let msg_type = msg_codec::get_msg_type(&message);
    if (msg_type == VANILLA_TYPE()) {
        increment_inbound_count(src_eid);
    } [...]
    else if (msg_type == ABA_TYPE()) {
        increment_inbound_count(src_eid);
        let options = new_empty_type_3_options();
        append_executor_options(&mut options, &new_executor_options(
            vector[
                new_lz_receive_option(2000000, 100),
            ],
            vector[],
            vector[],
            false,
        ));
        assert!(option::is_some(&receive_value), EABA_REQUIRES_RECEIVE_VALUE);
        increment_outbound_count(src_eid);
        let b_message = msg_codec::encode_msg_type(msg_codec::VANILLA_TYPE(),
            ↪ src_eid, option::some(10));
        let zro_fee = option::none();
        lz_send(src_eid, b_message, options, option::borrow_mut(&mut receive_value), &mut
            ↪ zro_fee);
        option::destroy_none(zro_fee);
    }
    [...]
}
```

Also, in the Move implementation, there is an inconsistency in the assignment of values to the message and executor options (10 and 100, respectively).

```
>_ evm/oapp/contracts/oapp/examples /OmniCounterAbstract.sol
```

SOLIDITY

```
// The EVM implementation
function _lzReceive(
    [...]
    internal override {
        [...]
        if (messageType == MsgCodec.VANILLA_TYPE) {
            count++;
            //////////////////////////////////// IMPORTANT ////////////////////////////////////
            /// if you request for msg.value in the options, you should also encode it
            /// into your message and check the value received at destination (example below).
            /// if not, the executor could potentially provide less msg.value than you requested
            /// leading to unintended behavior. Another option is to assert the executor to be
            /// one that you trust.
            ////////////////////////////////////
            require(msg.value >= _message.value(), "OmniCounter: insufficient value");
            _incrementInbound(_origin.srcEid);
        }
        [...]
    }
}
```

## Remediation

Ensure `counter::lz_receive_impl` asserts that `receive_value` equals the value encoded in the message, and also verify the values passed in the message and the executor options are consistent.

## Unverified Cross-Chain Messaging

OS-LZV-SUG-02

### Description

Any `OApp` may call `endpoint::send_compose` to send messages to another `OApp` and trigger their `lz_compose`. Since `lz_compose` is public, the target `OApp` will automatically receive and process the message. In contrast to `lz_receive`, which has built-in security checks such as validating the sender's identity, verifying message pathways, and ensuring the sender is a trusted peer, `lz_compose_impl` in `counter` does not verify the sender of the message. Consequently, in `counter`, it is possible to send a across-chain message to any `OApp`'s peers via `COMPOSED_ABA_TYPE`.

### Remediation

Ensure the receiving `OApp` implements validation mechanisms. Additionally, incorporate checks in `lz_receive` to ensure that incoming messages are from trusted sources.

### Patch

Resolved in [2a4f7cf](#).

## Burning of Nilified Packets

OS-LZV-SUG-03

### Description

The issue relates to the behavior of `channels::nilify`, where the comment claims that *"Nilified packets cannot be burnt"*. This statement is inaccurate based on the actual logic of the function. `nilify` sets the `payload_hash` of a message to a constant value, `NIL_PAYLOAD_HASH`, for packets with a nonce greater than the lazy inbound nonce or for packets that already have a payload hash assigned within a particular channel. To burn a packet, either its nonce must be less than or equal to the lazy inbound nonce, or its payload hash must exist.

```
> _ endpoint_v2/sources/internal/channels.move RUST

/// Marks a packet as verified but disables execution until it is re-verified
/// Nilified packets cannot be burnt.
public(friend) fun nilify(receiver: address, src_eid: u32, sender: Bytes32, nonce: u64,
    → payload_hash: Bytes32) {
    assert!(
        nonce > store::lazy_inbound_nonce(receiver, src_eid, sender) || store::has_payload_hash(
            receiver,
            src_eid,
            sender,
            nonce
        ),
        EINVALID_NONCE
    );
    let inbound_payload_hash = store::get_payload_hash(receiver, src_eid, sender, nonce);
    assert!(inbound_payload_hash == payload_hash, EPAYLOAD_HASH_DOES_NOT_MATCH);
    [...]
}
```

However, if `nilify` sets a packet's `payload_hash` to `NIL_PAYLOAD_HASH` and then, later, the lazy inbound nonce advances past this packet's nonce, the packet may theoretically be burned. This violates the assumption that nilified packets are immune to burning.

### Remediation

Include an additional check to ensure that the packet's payload hash is not equal to `NIL_PAYLOAD_HASH`. The burn function should check for `NIL_PAYLOAD_HASH` to prevent burning nilified packets.

### Patch

Resolved in [ba73fd6](#).

## Inconsistencies in DVN and Executor Modules

OS-LZV-SUG-04

### Description

1. Both the DVN and executor modules include `get_supported_option_types`, however, only the executor module provides a setter, `set_supported_option_types`. Since `supported_option_types` is initialized as an empty vector, `get_supported_option_types` in DVN consistently returns an empty vector. Furthermore, neither module utilizes the `supported_option_types` field.
2. Among the DVN and executor modules, only the executor module includes the `set_price_feed_delegate` setter. Since `has_price_fee_delegate` is invoked within `get_dvn_fee_internal` through `get_effective_price_feed`, the DVN module should also implement a price feed delegate setter.
3. Both the DVN and executor modules include `get_default_multiplier_bps`, however, only the executor module provides the `set_default_multiplier_bps` setter. Since `default_multiplier_bps` is initialized to zero, `get_default_multiplier_bps` in DVN will always return zero. Given that `get_default_multiplier_bps` is invoked in `get_dvn_fee_internal`, the DVN module should include a setter for this parameter. Additionally, the setter should validate that the multiplier value is reasonable before applying it.
4. `set_executor_dst_config` and `set_dvn_dst_config` should include validation to ensure that the set configuration values are reasonable.

### Remediation

Ensure that the DVN and executor modules are modified as stated above and that any unnecessary or unutilized code is removed.

### Patch

Issue #1, #2 and #3 resolved in [d189326](#).

## Missing Validation Logic

OS-LZV-SUG-05

### Description

1. In `executor`, the `role_admin` array is initialized with exactly one address when the executor is first set up. The role admin may add or remove other admins or role admins, including removing themselves. They also have the ability to pause the executor through `set_pause`. This implies that the `role_admin` has full control over the admin functionalities and may pause the executor, and remove all role admins including themselves, rendering the `executor` in a state where it is permanently paused with no admin to unpause it.
2. `set_enforced_options` should ensure that the passed `msg_type` has a valid value. Additionally, before updating the `enforced_options`, the function should check if the new options differ from the current ones. This will prevent redundant state changes and unnecessary emissions.

```
>_ oapps/oft/sources/internal_oapp/oapp_core.move
```

RUST

```
public entry fun set_enforced_options(  
    [...]  
) {  
    assert_admin(address_of(move account));  
    assert_options_type_3(enforced_options);  
    oapp_store::set_enforced_options(eid, msg_type, enforced_options);  
    emit(EnforcedOptionSet { eid, msg_type, enforced_options });  
}
```

3. In `oft_v1_msg_codec`, functions like `sender`, `compose_gas`, and `compose_message_content` directly attempt to extract compose-related fields from the message. Thus, it is assumed that the message has a compose section if invoked. However, this assumption may result in errors when a message does not include a compose section.
4. `unset_rate_limit` removes the rate limit entry for a given endpoint ID ( `eid` ) from the `rate_limit_by_eid` table utilizing `table::remove`. However, it does not validate whether the rate limit exists for the given `eid` before attempting to remove it.

```
>_ oft/sources/internal_oft/oft_impl_config.move
```

RUST

```
/// Unset the rate limit for a given EID  
public(friend) fun unset_rate_limit(eid: u32) acquires Config {  
    table::remove(&mut store_mut().rate_limit_by_eid, eid);  
    emit(RateLimitUnset { eid });  
}
```

## Remediation

1. Enforce a rule that there must always be at least one `role_admin` at all times.
2. Incorporate the above-stated validation into the codebase.
3. `oft_v1_msg_codec` should validate whether a message includes a compose section before performing compose-specific operations.
4. Utilize `has_rate_limit` to verify if a rate limit exists for a given `eid`.

Issue #4 resolved in [c158d6f](#).

## Code Maturity

OS-LZV-SUG-06

### Description

1. In `universal_config`, `is_zro` and `lock_zro_address` should utilize `is_zro_metadata` and `has_zro_metadata` respectively, to avoid code duplication. Also, if (`zro_address` == `@0`) check in `set_zro_address` could utilize `has_zro_metadata`.
2. `store::registered_msglibs` should utilize `store()`, and `store::get_default_send_library` and `store::get_default_receive_library` should utilize `has_default_send_library` and `has_default_receive_library`, respectively.
3. `store::get_receive_library_timeout` should utilize `has_receive_library_timeout` to improve error handling. Also, `get_default_receive_library_timeout` and `unset_default_receive_library_timeout` should utilize `has_default_receive_library_timeout`.
4. In `worker_config`, `assert_worker_admin` and `assert_worker_role_admin` should utilize `is_worker_admin` and `is_worker_role_admin` respectively. Additionally, `is_allowed` should utilize `allowList_contains` and `denyList_contains`.
5. `packet_v1_codec::is_valid_version` could utilize `get_version`, and `msglib_manager::register_library` should utilize `assert_library_registered`.
6. `worker_config_store::set_price_feed` could utilize `worker_store_mut`.
7. `configs_uln::from_bool` and `multisig_store::set_multisig_signers` should utilize different parameter names to avoid collision with Move keywords.
8. `OFT` should implement `lz_compose`, and `lz_receive_impl` should not be `public`.

### Remediation

Implement the above-mentioned suggestions.

### Patch

Issue #6 and `packet_v1_codec::is_valid_version` from issue #5 were resolved in [31c8e04](#) while issue #8 was resolved in [ae4791d](#).

## Code Clarity

OS-LZV-SUG-07

### Description

1. The comment inside `clear_payload` ("Make sure that all hashes are present up to the clear-to point, clear them, and update the lazy nonce") suggests clearing all nonces up to the "clear-to" point, but the actual loop only clears the current nonce, not the entire range.
2. Some modules have functions marked as `public(friend)`, although they are only utilized in test scenarios, such as in the endpoint module. These test functions may be simplified to regular `fun` and invoked with appropriate test wrappers, improving clarity and reducing unnecessary exposure.
3. In the current implementation of `contract_identity`, `ContractSigner` structure should not have the `drop` ability to avoid accidental dropping.

```
>_ endpoint_v2_common/sources/contract_identity.move RUST  
  
module endpoint_v2_common::contract_identity {  
  [...]   
  /// Struct to persist the contract identity  
  /// Access to the contract signer provides universal access to the contract's  
  //    ↪ authority and should be protected  
  struct ContractSigner has store, copy, drop { contract_address: address }  
  [...]   
}
```

4. `store::get_payload_hash` should not utilize `EUNREGISTERED_PATHWAY` error code while checking the hashes, as it may result in ambiguity regarding the source of the error.

### Remediation

1. Update the comment to reflect the fact that only the current nonce is cleared and clarify any related expectations about nonce management.
2. Utilize `fun` in functions, with test wrappers to call them from test scenarios.
3. Remove the `drop` ability from the `ContractSigner` structure.
4. Assign a distinct error code for each check to improve clarity.

### Patch

1. Issue #3 resolved in [59a30f8](#).
2. Issue #4 resolved in [dd97382](#).

## Removal of Duplicate Events

OS-LZV-SUG-08

### Description

There are multiple cases where a lack of proper checks results in setter functions being executed multiple times with the same inputs, leading to the emission of duplicate events.

1. `register_receive_pathway` does not verify whether the pathway has already been registered. It should check for an existing pathway registration and omit the event emission if it is already registered.
2. `set_peer` in OApps should restrict the registration of a peer multiple times to avoid duplicated events.
3. `universal_config::lock_zro_address` may lock an already locked store resulting in unnecessary event emission.
4. `enable_feed_updater` should check if the updater is already on the list before setting the feed updater.

```
>_ price_feed_modules/price_feed_module_0/sources/feeds.move
```

RUST

```
/// Gives a feed updater permission to write to the signer's feed
public entry fun enable_feed_updater(account: &signer, updater: address) acquires Feed {
    let feed_address = address_of(move account);
    table::upsert(feed_updaters_mut(feed_address), updater, true);
    emit(FeedUpdaterSet { feed_address, updater, enabled: true });
}
```

5. `set_deposit_address` should include a check to prevent setting the same address repeatedly.
6. Within `oapp_core`, `set_enforced_options` should check that it is not setting the same options, `transfer_admin` should check that it is not setting the same admin, and `set_delegate` should check that it is not setting the same delegate.
7. A packet may be nilified multiple times. `channels::nilify` should check if the payload hash is not `NIL_PAYLOAD_HASH` to deny an already nilified message.

### Remediation

Ensure that the functions add the checks to prevent the emission of duplicate events.

### Patch

Issue #3 resolved in [b612f0f](#).

## Code Refactoring

OS-LZV-SUG-09

### Description

1. Setter functions should avoid emitting events or allowing changes when there is no actual state change. Doing so can prevent unnecessary gas costs, reduce noise in logs, and ensure the system operates more efficiently, with events remaining meaningful.
2. `oft_impl_config::debit_view_with_possible_fee` currently utilizes a hardcoded value of `10_000` as the divisor for calculating the fee. If the system decides to change the scale, the function will become incompatible. Replace the hardcoded value with a `MAX_FEE_BPS` constant.

```
>_ oft/sources/internal_offt/oft_impl_config.move RUST

public(friend) fun debit_view_with_possible_fee(
    amount_ld: u64,
    min_amount_ld: u64,
): (u64, u64) acquires Config {
    [...]
    else {
        [...]
        // Calculate the preliminary fee based on the amount provided; this may increase
        //   ↳ when dust is added to it.
        // The actual fee is the amount sent - amount received, which is fee + dust
        //   ↳ removed
        let preliminary_fee = (amount_ld * fee_bps) / 10_000;
        [...]
    }
}
```

3. `oft_impl_config::in_flight_at_timestamp` could be optimized by adding an early return when the elapsed time is greater than or equal to the rate limit's window duration. This will improve code clarity and efficiency by avoiding unnecessary calculations.
4. Across the codebase, there are unutilized error codes declared, such as `oft_core::EUNAUTHORIZED` and `oapp_core::EUNEXPECTED_LZ_RECEIVE_VALUE`, which may be removed.

### Remediation

Incorporate the above-stated refactors.

## Error Handling

OS-LZV-SUG-10

### Description

1. The withdrawing functions should include a balance check before performing any withdrawal to ensure sufficient funds are available. This preemptive validation would allow for more graceful error handling, preventing unexpected failures.
2. `withdraw_coin` in `oft_coin` should verify that the account is registered for the specified `CoinType` before proceeding. This will provide better error handling and clearer feedback in cases of insufficient balance or an incorrect coin type.

```
>_ oft/sources/oft_implementation/oft_coin.move
```

RUST

```
/// Withdraw coin function abstracted from `oft.move` for cross-chain flexibility
public(friend) fun withdraw_coin<CoinType>(account: &signer, amount_ld: u64):
    ↪ Coin<CoinType> {
    coin::withdraw<CoinType>(account, amount_ld)
}
```

### Remediation

1. Add a balance check in the withdraw functions before performing a withdrawal.
2. Explicitly check account registration so that errors caused by unregistered accounts may be caught early and handled more gracefully.

## Separating Production and Test-specific Functionality

OS-LZV-SUG-11

### Description

The `*_at_timestamp` functions ( `try_consume_rate_limit_capacity_at_timestamp`, `in_flight_at_timestamp` ) in `oft_impl_config` are designed to operate with a timestamp argument. However, in production (non-test), they are consistently called with the current timestamp. The explicit inclusion of a timestamp argument, when always utilizing the current time, renders the code unnecessarily verbose and less intuitive. Due to the mix of test-driven functionality and production logic, the clarity and readability of the code are impacted.

### Remediation

Create separate test-specific functions that allow flexibility in passing arbitrary timestamps for testing purposes.

# A — Vulnerability Rating Scale

---

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the [General Findings](#).

---

## CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
  - Improperly designed economic incentives leading to loss of funds.
- 

## HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
  - Exploitation involving high capital requirement with respect to payout.
- 

## MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
  - Forced exceptions in the normal user flow.
- 

## LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.
- 

## INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
  - Improved input validation.
-

## B — Procedure

---

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.