OtterSec

Layerzero V2 SUI

Security Assessment

September 1st, 2025 — Prepared by OtterSec

Thiago Tavares thitav@osec.io

Michał Bochnak embe221ed@osec.io

Robert Chen r@osec.io

Table of Contents

01 — Executive Summary

Overview

Layerzero engaged OtterSec to assess the `sui` program. This assessment was conducted between August 12th and August 30th, 2025. A follow-up assessment was performed on September 30th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 5 findings throughout this audit engagement.

In particular, we identified a vulnerability where it is possible to enable fees without validating a proper fee recipient, which risks sending all collected fees to a zero address, resulting in revenue loss and accounting inconsistencies (OS-LVS-ADV-00). Additionally, we advised to explicitly add documentation to warn developers that call parameters are not guaranteed immutable, and also to add design-level improvement to allow configuration during call creation, deciding whether the callee should receive only an immutable or a mutable reference to the parameter (OS-LVS-ADV-01). Furthermore, we highlighted improper implementation of the multi-call logic (OS-LVS-ADV-02).

We also made recommendations for updating the codebase to ensure adherence to coding best practices (OS-LVS-SUG-01), and suggested including additional safety checks within the codebase to improve security (OS-LVS-SUG-00).

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 989c5f3. The follow-up audit was performed against 81c17a6.

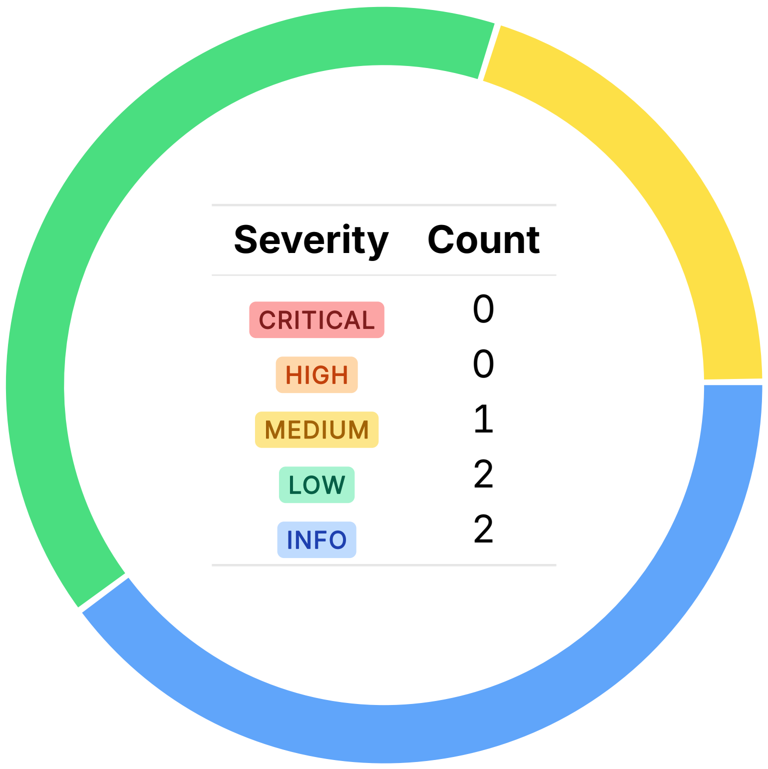
A brief description of the program is as follows:

Name	Description
sui	Layerzero on Sui introduces a Hot Potato execution-intent model to externalize routing, preserve modularity, and allow new message libraries without altering core contracts.

03 — Findings

Overall, we reported 5 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.



Severity	Count
CRITICAL	0
HIGH	0
MEDIUM	1
LOW	2
INFO	2

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-LVS-ADV-00	MEDIUM	RESOLVED ✓	Currently, it is possible to enable fees without validating a proper fee recipient, which risks sending all collected fees to a zero address, resulting in revenue loss and accounting inconsistencies.
OS-LVS-ADV-01	LOW	RESOLVED ✓	`Call` lets callees mutably borrow the `call` parameter, which may result in unintended modifications. While safe in the current trusted setup, documenting this or adding an option to restrict mutability would improve safety.
OS-LVS-ADV-02	LOW	RESOLVED ✓	The original `multi_call` implementation only supported unique callees, resulting in failures when the same DVN appeared multiple times in required and optional lists. Deduplication broke parameter alignment, resulting in to incorrect calls.

Missing Fee Recipient Validation MEDIUM OS-LVS-ADV-00

Description

`treasury::set_fee_enabled` allows enabling fees without validating that a proper `fee_recipient` (`!=@0x0`) is configured. If fees are turned on while the recipient remains unset (`@0x0`), all collected fees may be lost, effectively burning protocol revenue. This creates a silent failure where users pay fees, but the treasury is unable to collect them.

>_ *contracts/message-libs/treasury/sources/treasury.move* RUST

```
/// Enables or disables all fee collection globally.
///
/// When disabled, the treasury will return zero fees regardless of
/// other configuration settings. This is a master switch for all
/// fee collection functionality.
public fun set_fee_enabled(self: &mut Treasury, _admin: &AdminCap, fee_enabled: bool) {
    self.fee_enabled = fee_enabled;
    event::emit(FeeEnabledSetEvent { fee_enabled });
}
```

Remediation

Assert that the `fee_recepient` is set and valid before enabling fees.

Patch

Resolved in b43d3c8.

Call Parameter Mutability Risk LOW

OS-LVS-ADV-01

Description

`Call` allows the `callee` to mutably borrow the `call` parameter, which may result in unintended modifications that the `caller` does not anticipate. In `endpoint_v2::confirm_send`, after `destroy_child` is invoked, the parameter is still utilized when confirming the packet in the messaging channel. If the `callee` had altered it, this would affect correctness. While this is not a major risk here since the message library is a trusted, admin-registered party, it may be problematic in less controlled contexts.

Remediation

To prevent misuse, the documentation should explicitly warn developers that `call` parameters are not guaranteed immutable. A more explicit, design-level improvement will be to allow configuration during `call` creation, deciding whether the `callee` should receive only an immutable or a mutable reference to the parameter.

Patch

Resolved in de19d28.

Improper Multi Call Implementation LOW

OS-LVS-ADV-02

Description

The old `multi_call` implementation required all callees to be unique, so if the same callee appeared multiple times, only the first instance would be borrowed and executed. In ULN302, there are two DVN lists, required and optional, where the same DVN may appear in both, resulting in repeated callees in a `multi_call`, rendering it impossible for the DVNs to complete their calls. The initial patch attempted to deduplicate the DVNs to satisfy the old `multi_call` logic, but this shifted DVN indices, resulting in parameters to be sent to the wrong DVNs (since the parameters are sent by the DVN index, which may change after the deduplication).

Remediation

Update the logic to utilize `sequential_multi_call`, which supports repeated callees by specifying the borrow index for each call.

Patch

Resolved in 37fdcb0.

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-LVS-SUG-00	There are several instances where the code requires refactoring to ensure proper validation logic, to mitigate potential security risks, and to improve robustness.
OS-LVS-SUG-01	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.

Missing Validation Logic OS-LVS-SUG-00

Description

1. The setter (`set_*`) functions in `treasury` emit events even when the new value equals the current one, creating redundant events, which clutter the event logs and waste computation resources. Add assertions to prevent meaningless updates, ensuring events only reflect real changes.
2. The current `is_package` check in `call_cap` is fragile because it may misclassify future capability types. Explicitly checking `self.cap_type == CapType::Package` will be more accurate and future-proof against new `CapType` variants.

>_ *contracts/call/sources/call_cap.move* RUST

```rust
/// Check if this is a Package CallCap
public fun is_package(self: &CallCap): bool {
    self.cap_type != CapType::Individual
}
```

Remediation

Incorporate the validation checks stated above.

Patch

The issues were acknowledged by Layerzero.

Code Maturity

OS-LVS-SUG-01

Description

1. The comment in `endpoint_v2::quote` (*Using* `self.id()` *here is to ensure the Endpoint is initialized.*) incorrectly references `self.id()` instead of `self.eid()`, which is actually utilized in the code. Rectify the comment to reflect the code implementation.

   >_ *contractS/endpoint-v2/sources/endpoint_v2.move* RUST

   ```
   public fun quote([...]): Call<MessageLibQuoteParam, MessagingFee> {
       call.assert_caller(messaging_channel.oapp());
       let (send_lib, _) = self.message_lib_manager.get_send_library(call.caller(),
           ↪  call.param().dst_eid());
       // Create a outbound call to the message-lib to process the quote.
       // Using self.id() here is to ensure the Endpoint is initialized.
       let quote_param = messaging_channel.quote(self.eid(), call.param());
       call.create_single_child(&self.call_cap, send_lib, quote_param, ctx)
   }
   ```

2. `worker_options` currently utilizes all-uppercase error constants (`EINVALID_WORKER_ID`), which breaks consistency with other modules. Adopting `PascalCase` (`EInvalidWorkerId`) improves readability and aligns with established conventions in the codebase.
3. `CoinMetadata<ZRO>` should be frozen right after `coin::create_currency` to prevent later modification of `decimals`, `name`, or `symbol`, ensuring token details remain immutable
4. The `build_*` functions in `dvn::hashes` currently utilize manual vector appends, which are verbose and error-prone. Utilizing dot notation with a buffer writer will render serialization cleaner and more readable.

Remediation

Implement the above-mentioned suggestions.

Patch

Issue #1 and #2 were resolved in b43d3c8.

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.