OtterSec

Layerzero Endpoint

Security Assessment

December 22nd, 2025 — Prepared by OtterSec

James Wang james.wang@osec.io

Samuel Bétrisey sam@osec.io

Table of Contents

01 — Executive Summary

Overview

Layerzero engaged OtterSec to assess the `layerero-endpoint` program. This assessment was conducted between September 13th and October 9th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 4 findings throughout this audit engagement.

In particular, we made recommendations for modifying the codebase to improve consistency, functionality, and removal of unnecessary code, ensuring adherence to coding best practices (OS-LZE-SUG-02). We also highlighted several deviations from the solidity implementation of Layerzero, and advised aligning the StarkNet implementation with EVM (OS-LZE-SUG-03). Furthermore, we suggested incorporating proper validation logic to mitigate security issues (OS-LZE-SUG-01).

Scope

The source code was delivered to us in a Git repository at https://github.com/LayerZero-Labs/EPv2-Starknet. This audit was performed against commit 1e3fef4.

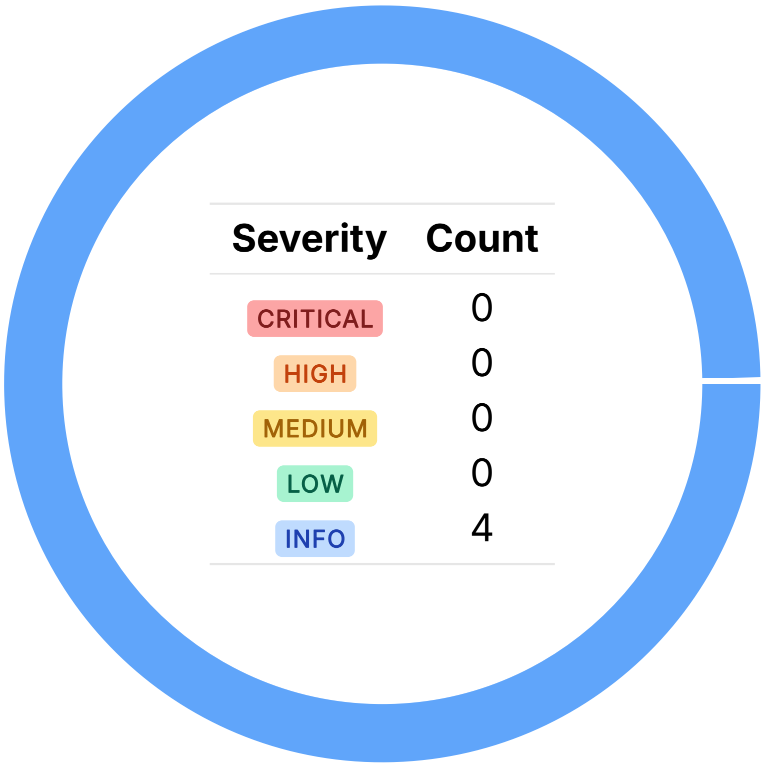
A brief description of the program is as follows:

Name	Description
layerero-endpoint	Adaptation of LayerZero’s cross-chain messaging protocol to Starknet. It manages message sending, verification by DVNs, and execution via Executors, while supporting upgradable workers, fee handling, and composable OApps and token interactions.

02 — Findings

Overall, we reported 4 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	0
LOW	0
INFO	4

03 — 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-LZE-SUG-00	StarkNet allows re-entrancy even in read-only calls, theoretically allowing the same nonce to be executed twice, as `commit` may utilize stale `lazy_nonce` data after `_initializable` reenters.
OS-LZE-SUG-01	There are several instances where proper validation is not performed, resulting in potential security issues.
OS-LZE-SUG-02	Recommendation for modifying the codebase to improve consistency, functionality, and removal of unnecessary code, ensuring adherence to coding best practices.
OS-LZE-SUG-03	There are multiple discrepancies where the Starknet implementation deviates from the EVM implementation.

Risk of Re-entrancy

OS-LZE-SUG-00

Description

StarkNet does not support true read-only calls, implying even query-like functions may be re-entrant. In `commit`, `_initializable` calls `receiver_dispatcher.allow_initialize_path`, which may reenter the endpoint and commit another payload.

>_ *layerzero/src/endpoint/endpoint.cairo* CAIRO

```cairo
fn _initializable(
    self: @ContractState,
    origin: Origin,
    receiver: ContractAddress,
    lazy_inbound_nonce: u64,
) -> bool {
    let receiver_dispatcher = ILayerZeroReceiverDispatcher { contract_address: receiver };
    receiver_dispatcher.allow_initialize_path(origin) || lazy_inbound_nonce > 0
}

fn commit(
    ref self: ContractState,
    origin: Origin,
    receiver: ContractAddress,
    payload_hash: Bytes32,
) {
    [...]
    // Ensure that the payload hash is valid
    assert_with_byte_array(payload_hash != EMPTY_PAYLOAD_HASH, err_invalid_payload_hash());
    [...]
}
```

This may update the `lazy_inbound_nonce` and clear the payload hash entry while the original commit is still running. Since `lazy_nonce` is not refetched after the external call, the later `_committable` check may incorrectly succeed with stale data. This theoretically breaks the guarantee that a nonce cannot be executed more than once. However, practically, the impact is limited, as only the `OApp` may exploit this against itself.

Remediation

Ensure to add re-entrancy protection.

Patch

Resolved in 6e4b284.

Missing Validation Logic OS-LZE-SUG-01

Description

1. `ultra_light_node::verifiable` does not check if `payload_hash` is empty, which may result in it returning incorrect verification states for non-existent messages. Ensure that `payload_hash` is not empty.
2. `options::split_options` silently ignores unknown `worker_ids`. Revert on any unrecognized `worker_id` to ensure only valid executor or DVN options are accepted.
3. `_pay_workers` only checks the sender's `ERC20` allowance before performing `transfer_from` calls to pay workers. Thus, if the sender's actual balance is insufficient, the transfer will panic with a generic error instead of the intended custom `err_insufficient_fee`. This behavior violates the requirement of throwing a custom error when fees are insufficient. Validate balances before transfers or utilize a `SafeDispatcher` to catch failures and return a custom error as implemented in the Executor.

>_ *layerzero/src/endpoint/endpoint.cairo* CAIRO

```
fn _pay_workers([...]) {
    // In order to satisfy the requirement of throwing a custom error in case the user
    // didn't send enough fees to pay all of the workers,
    [...]
    Self::_assert_messaging_fee(
        total_native_fee, native_allowance, total_zro_fee, zro_allowance,
    );
    [...]
}
```

Remediation

Add the missing validations.

Patch

1. Issue #1 resolved in 7d68f89.
2. Issue #2 resolved in 937c50d.
3. Issue #3 resolved in ca8b362.

Code Refactoring OS-LZE-SUG-02

Description

1. In the Executor, the `.expect("Alert event emitted")` message in `_execute` and `_compose` is misleading. If the alert emission fails, the contract panics with the string *"Alert event emitted"*, giving the false impression that the event was successfully emitted. This creates ambiguity when interpreting logs and obscures the actual cause of failure. Updating the message to reflect failure.
2. The project currently utilizes compiler version `2.12.0`, but a newer release (`2.12.2`) is available with bug fixes and improvements. Utilizing an older version may expose the system to known issues or require extra effort to confirm safety. Upgrade to the newer version to ensure compatibility and security.
3. `message_lib_manager` events may have been copied into endpoint events mistakenly. Ensure to remove those events to avoid confusion.
4. The re-entrancy component in `dvn::Event` is unutilized and may be removed.

>_ *layerzero/src/workers/dvn/dvn.cairo* CAIRO

```cairo
#[event]
#[derive(Drop, starknet::Event)]
pub enum Event {
    [....]
    #[flat]
    ReentrancyGuardEvent: ReentrancyGuardComponent::Event,
    [...]
}
```

Remediation

Incorporate the above refactors.

Patch

1. Issue #1 resolved in f94e2f8.
2. Issue #3 resolved in d4499ed.
3. Issue #4 resolved in 449f8e1.

Deviation from EVM Implementations OS-LZE-SUG-03

Description

1. In the StarkNet implementation `initializable` reverts in `ultra_light_node::verifiable` if the `OApp` call fails, while in EVM it is wrapped in `try/catch` and simply returns false. This creates inconsistent behavior across chains. Align Starknet's implementation with EVM by handling failures gracefully.
2. `ultra_light_node::set_default_executor_config` in Starknet only allows updating one executor configuration per call, whereas the EVM implementation in `SendLibBase::setDefaultExecutorConfigs` supports batch updates in a single transaction. This difference implies that setting multiple endpoints on Starknet requires multiple calls.

   >_ *layerzero/src/message_lib/uln/ultra_light_node.cairo* CAIRO

   ```
   fn set_default_executor_config(ref self: ContractState, dst_eid: u32, config:
       ↪  ExecutorConfig,) {
       self.ownable.assert_only_owner();
       self.executor_configs.entry(DEFAULT_CONFIG).entry(dst_eid).write(config.clone());
       self.emit(DefaultExecutorConfigSet { dst_eid, config });
   }
   ```

3. In Starknet's `dvn_fee_lib`, `_get_calldata_size` utilizes incorrect constants, where `EXECUTE_FIXED_BYTES` is 68 and `VERIFY_BYTES` is 320, rather than utilizing the correct values of 260 and 288, respectively, as done in `DVNFeeLib` in EVM.
4. In the Starknet implementation, within `oapp_core`, the `initializer` only stores the endpoint and native token but does not set a delegate. Unlike the EVM constructor, which enforces delegate assignment. Ensure to assign delegators for `oapp` on initialization in `oapp_core`.

   >_ *layerzero/src/oapps/oapp/oapp_core.cairo* CAIRO

   ```
   fn initializer(
       ref self: ComponentState<TContractState>,
       endpoint: ContractAddress,
       native_token: ContractAddress,
   ) {
       self.OAppCore_endpoint.write(endpoint);
       self.OAppCore_native_token.write(native_token);
   }
   ```

Remediation

Ensure the StarkNet implementation aligns with EVM for better consistency.

Patch

1. Issue #1 resolved in 7d68f89.
2. Issue #2 resolved in 5dacaf5.
3. Issue #3 resolved in e1b0d0a.
4. Issue #4 resolved in e9e3100.

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.