AstraSec

ULAB
Security Audit Report

December 19, 2025

Contents

1 Introduction

1.1 About ULAB

ULAB is a cross-chain token bridge developed by **LayerBank**, built on LayerZero V2's Omnichain Fungible Token (OFT) standard. The protocol enables seamless ULAB token transfers across multiple blockchain networks using a burn-and-mint mechanism—tokens are burned on the source chain and minted on the destination chain, preserving total supply integrity. The bridge incorporates rate limiting to control transfer volume, access control via whitelist and blacklist, and a configurable fee system.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/layerbank-foundation/ulab-oft
- Commit: 27529f6

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/layerbank-foundation/ulab-oft
- Commit: ebcf08c

1.3 Revision History

Version	Date	Description
v1.0	December 19, 2025	Initial Audit

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the ULAB protocol. Throughout this audit, we identified a total of 4 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	–	–	–	–
High	–	–	–	–
Medium	3	1	–	2
Low	1	–	–	1
Informational	–	–	–	–

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~M-1~~	Missing Slippage Protection in ulab.move
~~M-2~~	Potential DoS Due to Multiplication Overflow
M-3	Potential Risks Associated with Centralization
~~L-1~~	Inconsistent Rate Limit Behavior Between Aptos and EVM

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

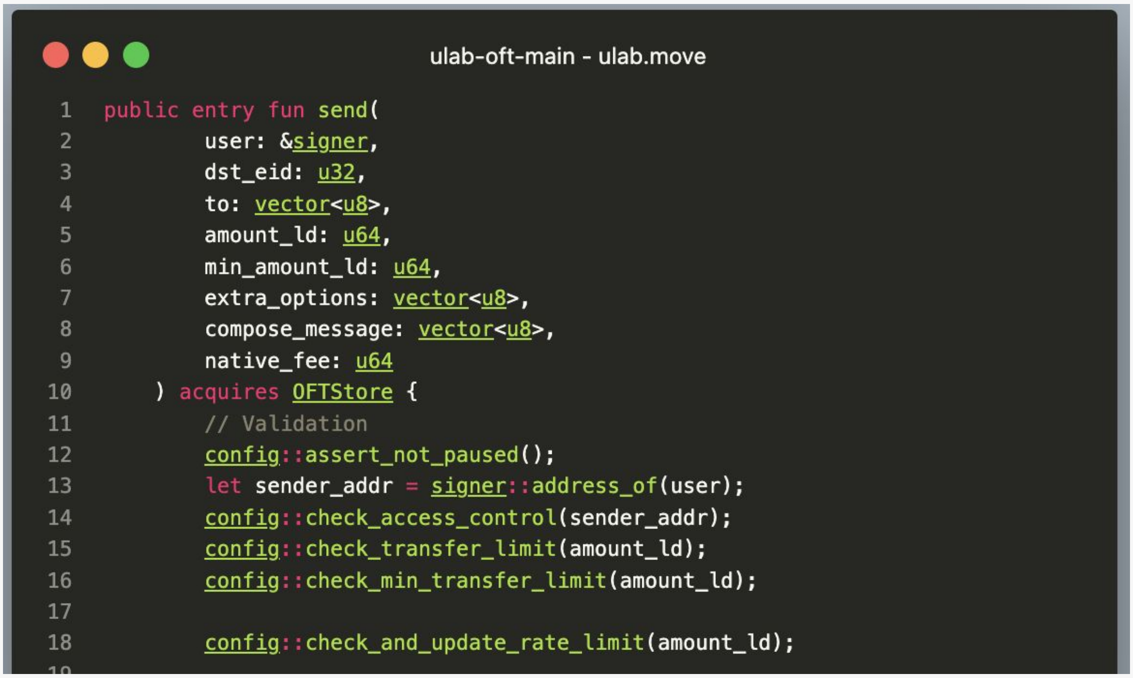
Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and re-mediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

3.3.1 [M-1] Missing Slippage Protection in ulab.move

Target	Category	IMPACT	LIKELIHOOD	STATUS
ulab.move	Business Logic	Medium	Medium	Addressed

The send() function accepts a min_amount_ld parameter intended for slippage protection, but the parameter is never used throughout the function. Users may receive significantly less tokens than expected due to bridge fees and precision loss during decimal conversion (8 → 6 decimals), with no mechanism to revert the transaction if the received amount falls below their acceptable threshold.


ulab-oft-main - ulab.move

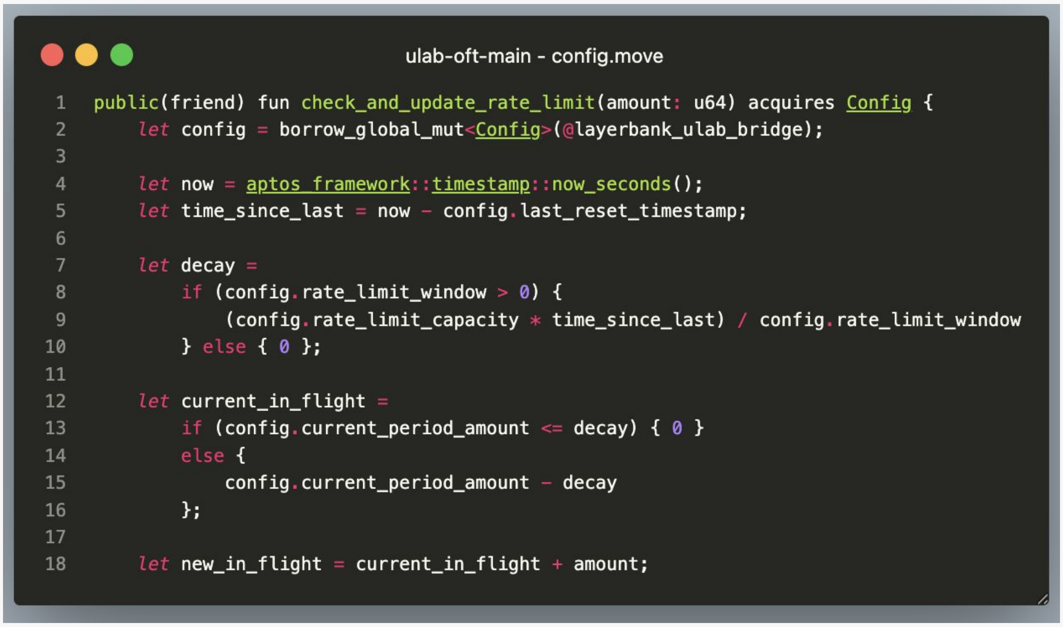
```
public entry fun send(
        user: &signer,
        dst_eid: u32,
        to: vector<u8>,
        amount_ld: u64,
        min_amount_ld: u64,
        extra_options: vector<u8>,
        compose_message: vector<u8>,
        native_fee: u64
    ) acquires OFTStore {
        // Validation
        config::assert_not_paused();
        let sender_addr = signer::address_of(user);
        config::check_access_control(sender_addr);
        config::check_transfer_limit(amount_ld);
        config::check_min_transfer_limit(amount_ld);

        config::check_and_update_rate_limit(amount_ld);
```

Remediation Add a slippage check before burning tokens to ensure the actual bridged amount meets the user's minimum expectation.

3.3.2 [M-2] Potential DoS Due to Multiplication Overflow

Target	Category	IMPACT	LIKELIHOOD	STATUS
config.move	Business Logic	Medium	Medium	Addressed

The rate limit decay calculation (config.rate_limit_capacity * time_since_last) / config.rate_limit_window performs multiplication before division using u64 arithmetic. If the bridge remains inactive for an extended period (approximately 21 days with a typical capacity of 10 billion tokens at 8 decimals), the multiplication can overflow u64::MAX (~1.8 × 10^{19}), causing the transaction to revert and effectively blocking all bridge operations until an admin intervenes.



```
public(friend) fun check_and_update_rate_limit(amount: u64) acquires Config {
    let config = borrow_global_mut<Config>(@layerbank_ulab_bridge);

    let now = aptos_framework::timestamp::now_seconds();
    let time_since_last = now - config.last_reset_timestamp;

    let decay =
        if (config.rate_limit_window > 0) {
            (config.rate_limit_capacity * time_since_last) / config.rate_limit_window
        } else { 0 };

    let current_in_flight =
        if (config.current_period_amount <= decay) { 0 }
        else {
            config.current_period_amount - decay
        };

    let new_in_flight = current_in_flight + amount;
```

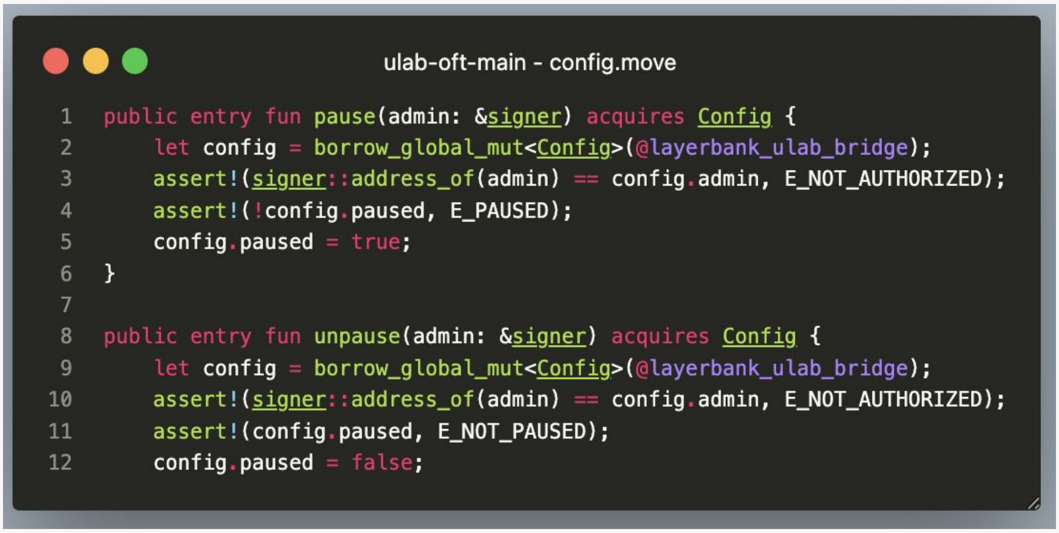
Remediation Use u128 for intermediate calculation to prevent overflow.

3.3.3 [M-3] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	High	Low	Acknowledged

The ULAB protocol relies on admin account to control over critical operations, introducing notable centralization risks. The admin account, assigned distinct roles, can unilaterally influence the protocol's functionality and integrity. Below are key examples of privileged functions and their associated roles:

- ➔ Rate Limit Configuration: Admin can modify the rate limit capacity and time window parameters at any time.
- ➔ Access Control Management: Admin can enable or disable whitelist/blacklist modes, and add or remove addresses from either list.
- ➔ Pause Control: Admin can pause and unpause all bridge operations.
- ➔ Peer Management: Admin can register trusted peer addresses on remote chains.
- ➔ LayerZero Configuration: Admin can set the message library and DVN/Executor configurations for cross-chain messaging.


ulab-oft-main - config.move

```
public entry fun pause(admin: &signer) acquires Config {
    let config = borrow_global_mut<Config>(@layerbank_ulab_bridge);
    assert!(signer::address_of(admin) == config.admin, E_NOT_AUTHORIZED);
    assert!(!config.paused, E_PAUSED);
    config.paused = true;
}

public entry fun unpause(admin: &signer) acquires Config {
    let config = borrow_global_mut<Config>(@layerbank_ulab_bridge);
    assert!(signer::address_of(admin) == config.admin, E_NOT_AUTHORIZED);
    assert!(config.paused, E_NOT_PAUSED);
    config.paused = false;
```

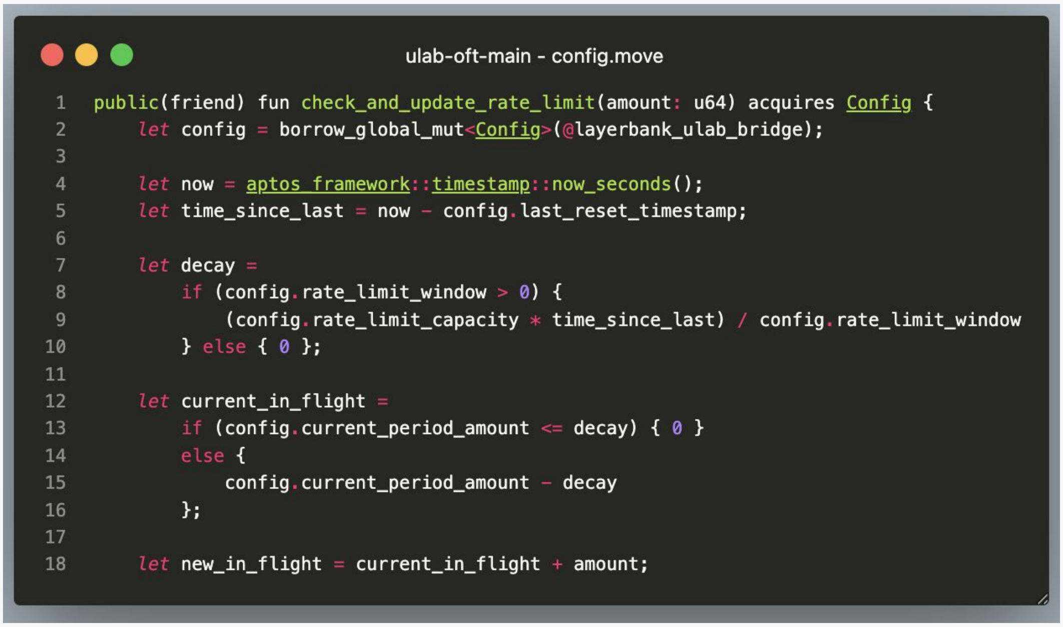
Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team This issue has been confirmed by the team.

3.3.4 [L-1] Inconsistent Rate Limit Behavior Between Aptos and EVM

Target	Category	IMPACT	LIKELIHOOD	STATUS
config.move	Business Logic	Low	Medium	Addressed

The Aptos and EVM implementations handle the rate_limit_window == 0 edge case in completely opposite ways. In Aptos, when the window is zero, the decay is set to zero, meaning the rate limit never resets and the bridge becomes permanently locked after the first transfer exhausts the capacity. In contrast, the EVM implementation uses a divisor of 1 when the window is zero, causing the rate limit to decay almost instantly and effectively disabling rate limiting entirely. This inconsistency creates unpredictable cross-chain behavior for the same configuration



```
public(friend) fun check_and_update_rate_limit(amount: u64) acquires Config {
    let config = borrow_global_mut<Config>(@layerbank_ulab_bridge);

    let now = aptos_framework::timestamp::now_seconds();
    let time_since_last = now - config.last_reset_timestamp;

    let decay =
        if (config.rate_limit_window > 0) {
            (config.rate_limit_capacity * time_since_last) / config.rate_limit_window
        } else { 0 };

    let current_in_flight =
        if (config.current_period_amount <= decay) { 0 }
        else {
            config.current_period_amount - decay
        };

    let new_in_flight = current_in_flight + amount;
```

Remediation Add an early return check when rate limiting is not configured (window or capacity is zero) to provide consistent and predictable behavior across both implementations.

4 Appendix

4.1 About AstraSec

AstraSec is a premier blockchain security firm dedicated to delivering high-quality auditing services for blockchain-based protocols. Composed of veteran security researchers with extensive experience across the DeFi landscape, our team maintains an unwavering commitment to excellence and precision. AstraSec’s comprehensive methodology and deep understanding of blockchain architecture enable us to ensure protocol resilience, making us a trusted partner for our clients.

4.2 Disclaimer

The content of this audit report is for informational purposes only and does not constitute legal, financial, or investment advice. The findings, views, and conclusions presented herein are based on the code and documentation provided at the specific time of the audit and may be subject to risks and uncertainties not detected during the assessment.

While AstraSec exerts every effort to ensure the accuracy and completeness of this report, the information is provided on an "as is" basis. We assume no responsibility for any errors, omissions, or inaccuracies. Users should conduct their own independent due diligence and consult with professional advisors before making any investment or deployment decisions. AstraSec shall not be held liable for any direct or indirect losses, damages, or consequences arising from reliance on this report.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://x.com/AstraSecAI