OtterSec

Stargate V2 EndpointV2Alt Support

Security Assessment

February 5th, 2026 — Prepared by OtterSec

Nicholas R. Putra nicholas@osec.io

Robert Chen r@osec.io

Table of Contents

01 — Executive Summary

Overview

LayerZero engaged OtterSec to assess the `stargateV2-EndpointV2Alt` program. This assessment was conducted on October 21st, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

In particular, we identified a vulnerability in the token ownership transfer function, which lacks access control, allowing anyone to take over the TIP-20 admin role (OS-SES-ADV-00). We also made suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices (OS-SES-SUG-00).

Scope

The source code was delivered to us in a Git repository at https://github.com/stargate-protocol/stargate-v2. This audit was performed against 5ee7c7c.

A brief descriptions of the program is as follows:

Name	Description
stargateV2-EndpointV2Alt	The scope introduces ALT-endpoint variants of Stargate V2 contracts, which enforce ERC-20 token–based messaging fees, disallow native ETH, support taxi-only execution, and ensure compatibility with ALT endpoints and TIP-20 while preserving existing Stargate invariants and execution flows.

02 — 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-SES-ADV-00	MEDIUM	RESOLVED ✓	`transferTokenOwnership` currently lacks access control, allowing anyone to take over the TIP-20 admin role.

Unrestricted TIP-20 Ownership Transfer Functinoality MEDIUM OS-SES-ADV-00

Description

`transferTokenOwnership` in `StargateOFTTIP20` currently lacks access control, allowing anyone to call it. This lets an attacker grant themselves the TIP-20 admin role, gaining the ability to mint, burn, or manipulate the token.

>_ *packages/stg-evm-v2/src/tip20/StargateOFTTIP20.sol* SOLIDITY

```solidity
/// @notice Transfer the TIP-20 admin role to a new owner.
/// @dev Grants DEFAULT_ADMIN_ROLE to `_newOwner` and renounces it for the current admin.
/// @dev It mimics the transfer ownership functionality for the ERC20 tokens with roles.
/// @param _newOwner The account to receive the admin role.
function transferTokenOwnership(address _newOwner) external virtual override {
    // grant the role to the new owner and renounce it to remove it from current
    ITIP20RolesAuth(token).grantRole(DEFAULT_ADMIN_ROLE, _newOwner);
    ITIP20RolesAuth(token).renounceRole(DEFAULT_ADMIN_ROLE);
}
```

Remediation

Ensure `transferTokenOwnership` is protected yet with the `onlyOwner` modifier.

Patch

Resolved in c647d82.

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-SES-SUG-00	Suggestions to improve consistency, safety, and adherence to best practices across the codebase.

Code Maturity

OS-SES-SUG-00

Description

1. In `StargateOFTTIP20`, add a check to ensure `_newOwner` in `transferTokenOwnership` is not the zero address.
2. In `TokenMessagingAlt`, the constructor lacks a check to ensure the endpoint's `nativeToken` is a valid ERC20 token.
3. In `TokenMessagingAlt`, disable bus-related functions (e.g., `rideBus`, `driveBus`, `setFares`) since bus mode is not supported on Alt chains.
4. In `StargateOFTAlt`, override `recoverToken` to prevent the `treasurer` from recovering native tokens in form of `ERC20` token.
5. In `StargateOFTAlt`, consider overriding `plannerFee` and `withdrawPlannerFee`, as the current implementation relies on `address(this).balance`, which is not applicable on Alt chains where the native fee token is an ERC20.

Remediation

Implement the above-mentioned suggestions.

Patch

Issue #1 was resolved in d4ccd6f and the remaining issues are acknowledged.

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.