By
Quit

Contract Review
Issue date
9/3/2025

Overview

The following is a review of the Send and Receive smart contract suite developed by Transient Labs, a system designed to facilitate token redemption mechanics where users send ERC-1155 tokens and receive various assets in return. The contracts implement four redemption patterns: currency payouts, ERC-721 transfers, ERC-1155TL minting, and raffle-based distributions with provably fair randomness.

Contracts in scope for this review include:

- `src/SendAndReceiveCurrency.sol`
- `src/SendAndReceiveERC721.sol`
- `src/SendAndReceiveERC1155TL.sol`
- `src/SendAndReceiveERC1155TLRaffle.sol`
- `src/lib/SendAndReceiveBase.sol`
- `src/lib/AffinePermutation.sol`

This review is based on commit `c3835f266a79e5db0ed4d72c93d68d87b4efa0a0` and aims to identify security vulnerabilities, opportunities for gas optimization, and general best-practice recommendations with regard to the contracts in scope. The review should not be considered an endorsement of the project, nor is it a guarantee of security.

Note: This report has been updated to include remediations. It has been re-issued as of 9/4/2025 and includes commits up to `7d1ab9c60fb994b0a4704852b5095085950808d4`.

Findings

I-01: Base contract ERC-1155 receiver functions lack input amount validation

Severity: Informational

Status: Acknowledged

`onERC1155Received` and `onERC1155BatchReceived` in `SendAndReceiveBase` accept any token amounts without validation, relying on child contracts to implement amount checking within their `_processInputToken` functions. This creates a gap where invalid amounts reach the business logic layer before being rejected, and places the burden of validation on each child contract. While all current child contracts do implement this validation correctly, the base contract's receiver functions could provide stronger guarantees by performing amount validation before calling `_processInputToken`. Recommend adding an abstract function `_getRequiredInputAmount(address contractAddress, uint256 tokenId)` that

child contracts must implement, then performing amount validation in the base receiver functions.

I-02: Confusing boolean state names
Severity: Informational
Status: Resolved (`7d1ab9c60fb994b0a4704852b5095085950808d4`)
The contract uses `open` and `closed` boolean flags to track redemption state, creating semantics where a redemption can be simultaneously "open" and "closed" when finalized. The three possible states are: not started (`!open && !closed`), active (`open && !closed`), and finalized (`open && closed`). Recommend renaming the `closed` flag to `finalized` or `ended`, or using an enum with `NOT_STARTED`, `OPEN`, and `ENDED` entries to track state.

I-03: Documentation contains spelling errors
Severity: Informational
Status: Resolved (`7d1ab9c60fb994b0a4704852b5095085950808d4`)
`SendAndReceiveERC1155TL.sol` 1L6 contains "cnfigure" instead of "configure" and "redepmtion" instead of "redemption" in the contract's main description comment. AffinePermutation L4 reads “Permuation” instead of “Permutation”. Recommend correcting the spelling errors.

G-01: Struct packing can reduce storage costs
Severity: Gas Optimization
Status: Acknowledged
Structs throughout the contracts use inefficient storage layouts. For example, in `SendAndReceiveERC721.InputCofig`, If token amounts would fit into a uint96, it would pack with `contractAddress` into one slot. `SendAndReceiveERC721.Settings` uses uint64 values for `openAt` and `duration` which could safely be reduced to uint40, allowing these fields plus both boolean flags to pack with `tokenOwner` in a single slot. Other structs share similar optimization opportunities. Recommend restructuring all structs to pack smaller values with addresses.

G-02: Redundant timestamp checks in updateSettings function
Severity: Gas Optimization
Status: Resolved (`7d1ab9c60fb994b0a4704852b5095085950808d4`)
`updateSettings` in both `SendAndReceiveERC1155TL` and `SendAndReceiveERC721` contracts performs redundant `block.timestamp >= s.openAt` checks across multiple validation statements. The logic could be optimized by checking the timestamp condition once and then performing the parameter-specific validations within that branch. Recommend restructuring the validation logic to check `block.timestamp >= s.openAt` once, then conditionally validate individual parameter changes within that block.

L-01: Owner can withdraw user deposits before opening redemption
Severity: Low

Status: Resolved (`7d1ab9c60fb994b0a4704852b5095085950808d4`)
`withdrawCurrency` only restricts withdrawals of the configured currency during active redemptions, allowing the owner to withdraw user deposits before calling `open()`. Owners can immediately withdraw these deposits since the restriction only applies after redemption opens. This creates significant centralization risk where users fund the contract but receive no guarantee their funds will remain available. Recommend adding restrictions to prevent withdrawal of the configured currency before redemption opens.

L-02: ERC-721 not escrowed creates avoidable trust dependencies
Severity: Low
Status: Resolved (`7d1ab9c60fb994b0a4704852b5095085950808d4`)
Note: contract will be deployed via create2, allowing pre-approving the address. The contract will now pull the NFT upon deployment.
The contract transfers the ERC-721 directly from the configured `tokenOwner` to the recipient rather than escrowing the NFT in the contract itself. This requires users' trust that the owner will maintain approval and not transfer or revoke approval for the NFT during the redemption period. This trust assumption is documented in both the README and contract comments, however a more trust-minimized design would escrow the NFT in the contract during initialization, removing the owner's ability to interfere with redemptions once the event begins.

Summary

The codebase shows solid security fundamentals with consistent validation patterns across contracts. The base contract architecture works well, though centralizing input amount validation (I-01) would eliminate code duplication and reduce implementation risk. Most findings are informational or minor optimizations - struct packing opportunities (G-01, G-02) could save small amounts of gas.

The main trust model considerations involve pre-funding protection in the currency contract (L-01) and the NFT approval dependency in the ERC-721 version (L-02). Both are documented behaviors but represent areas where additional user protections could reduce centralization risk. The raffle implementation is mathematically sound with the `AffinePermutation` library providing impressively efficient winner selection. Overall, the contracts achieve their intended functionality with reasonable security, and the findings primarily focus on polish and optimization opportunities rather than critical vulnerabilities.

There may be an opportunity to consolidate the four `SendAndReceive` contracts into a single configurable implementation. The contracts share many code patterns, and a unified contract with redemption type configuration could reduce code duplication and simplify maintenance, while preserving the same functionality through logic branching for different redemption types (and perhaps an extension for raffle redemptions).