Wormhole NTT Solana

Security Assessment

May 9th, 2025 — Prepared by OtterSec

Akash Gurugunti sud0u53r.ak@osec.io

Robert Chen notdeghost@osec.io

Table of Contents

01 — Executive Summary

Overview

Wormhole Foundation engaged OtterSec to assess the `example-native-token-transfers` program. This assessment was conducted between April 22nd and May 5th, 2025. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 3 findings throughout this audit engagement.

In particular, we provided recommendations to ensure adherence to coding best practices (OS-WNS-SUG-02) and suggested implementing proper validations and explicit checks to improve overall security (OS-WNS-SUG-01). We further highlighted the underestimation of the actual serialized size for several structures (OS-WNS-SUG-00).

Scope

The source code was delivered to us in a Git repository at https://github.com/wormholelabs-xyz/native-token-transfers. This audit was performed against commit 5f2882e.

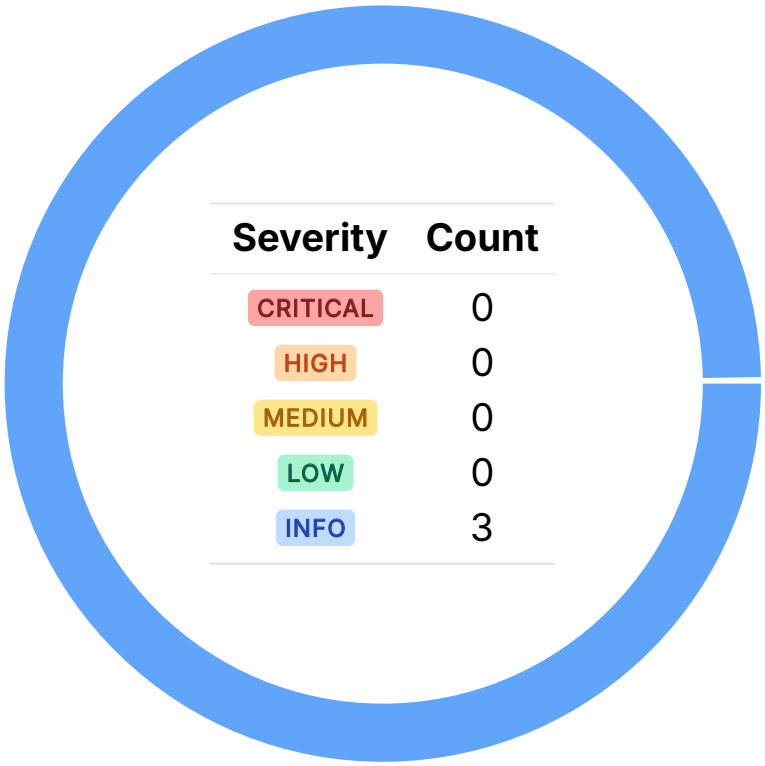
A brief description of the program is as follows:

Name	Description
example-native-token-transfers	It enables secure, modular cross-chain transfers of native tokens on Solana. It separates token transfer logic (NTT Manager) from message transport (Transceiver, e.g., Wormhole) using a shared interface (ntt-common).

02 — Findings

Overall, we reported 3 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	3

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-WNS-SUG-00	`written_size` under-reports the actual serialized size for several structs, resulting in unnecessary vector reallocations.
OS-WNS-SUG-01	There are several instances where proper validation is not performed, resulting in potential security issues.
OS-WNS-SUG-02	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices and removing .

Serialization Size Mismatch OS-WNS-SUG-00

Description

There are multiple instances where the size returned by `written_size` fails to match the actual data written, underestimating the actual serialized size. In `TransceiverMessage`, it omits the size of `recipient_ntt_manager` and the length-prefixed `transceiver_payload`. For both `WormholeTransceiverInfo` and `WormholeTransceiverRegistration`, it fails to include 4 bytes utilized for the `PREFIX`.

Remediation

Ensure `written_size` precisely mirrors all serialized components.

Patch

Fixed in PR#621.

Missing Validation Logic OS-WNS-SUG-01

Description

1. `set_token_authority` always updates the `rent_payer` field, even if the `pending_token_authority` account is already initialized. When the account is eventually closed, the lamports (SOL) it holds are refunded to the `rent_payer`. Thus, if this instruction is called again after initialization with a new `rent_payer`, the new rent payer will receive this refund without ever having funded the account. Add a check to ensure only `pending_authority` is updated after initialization and avoid modifying `rent_payer`.

   >_ *src/instructions/admin/transfer_token_authority.rs* RUST

   ```rust
   pub fn set_token_authority(ctx: Context<SetTokenAuthorityChecked>) -> Result<()> {
       ctx.accounts
           .pending_token_authority
           .set_inner(PendingTokenAuthority {
               bump: ctx.bumps.pending_token_authority,
               pending_authority: ctx.accounts.common.new_authority.key(),
               rent_payer: ctx.accounts.rent_payer.key(),
           });
       Ok(())
   }
   ```

2. In `set_peer`, should validate that `SetPeerArgs.chain_id` is not equal to the local `config.chain_id` to prevent a misconfiguration where a chain registers itself as a peer.
3. In `initialize_lut`, the `fee_collector` account in `WormholeAccounts` is added to the LUT without validation, creating a risk of including an incorrect account. Validate `fee_collector` to ensures only the correct account is added to the LUT.
4. In `AcceptTokenAuthorityBase`, `config.paused` check only restricts authority transfers within the program, but it will not prevent the current mint authority from calling the SPL token program directly to transfer authority to the PDA/multisig, rendering the pause mechanism ineffective at enforcing mint control externally.

Remediation

Incorporate the validations into the codebase.

Patch

1. Fixed in PR#614.
2. Fixed in PR#615.
3. Acknowledged.
4. Acknowledged.

Code Maturity OS-WNS-SUG-02

Description

1. As stated in the TODO, instead of writing separate `#[cfg(all(...))]` checks for each pair of mutually exclusive features, they may be combined into a single `#[cfg(any(...))]` block (as shown below) to simplify the code and improving maintainability.

>_ src/lib.rs RUST

```rust
#[cfg(any(
    all(feature = "mainnet", feature = "solana-devnet"),
    all(feature = "mainnet", feature = "tilt-devnet"),
    all(feature = "solana-devnet", feature = "tilt-devnet"),
))]
compile_error!(
    "features mainnet, solana-devnet and tilt-devnet are mutually exclusive"
);
```

2. In `transfer_owenership`, skipping the CPI call to `set_upgrade_authority_checked` if the current authority is already `upgrade_lock` avoids redundant operations, and simplifies the ownership transfer logic.
3. The `transceiver` account is redundant in `DeregisterTransceiver` and may be removed because its key is already stored in `registered_transceiver.transceiver_address` and may be utilized directly for PDA derivation.

Remediation

Implement the above-mentioned suggestions.

Patch

1. Acknowledged.
2. Fixed in PR#617.
3. Fixed in PR#616.

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.