Orderly Network

Security Assessment

November 22nd, 2024 — Prepared by OtterSec

Akash Gurugunti sud0u53r.ak@osec.io

Nicholas R. Putra sud0u53r.ak@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Orderly Network engaged OtterSec to assess the `soc-cc` and `solana-vault` programs. This assessment was conducted between November 4th and November 14th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 4 findings throughout this audit engagement.

We also made recommendations to ensure adherence to coding best practices (OS-OLN-SUG-02) and suggested enhancing the admin transfer process by utilizing a two-step verification process to confirm a change in the admin authority and removing the `TODO` comment (OS-OLN-SUG-00). We further advised to restrict updates to the order delivery setting after deployment, to avoid the risk of losing unexecuted messages when switching from unordered to ordered execution (OS-OLN-SUG-01).

Scope

The source code was delivered to us in a Git repository at https://github.com/OrderlyNetwork. This audit was performed against commits d0bbdda and ee16d85.

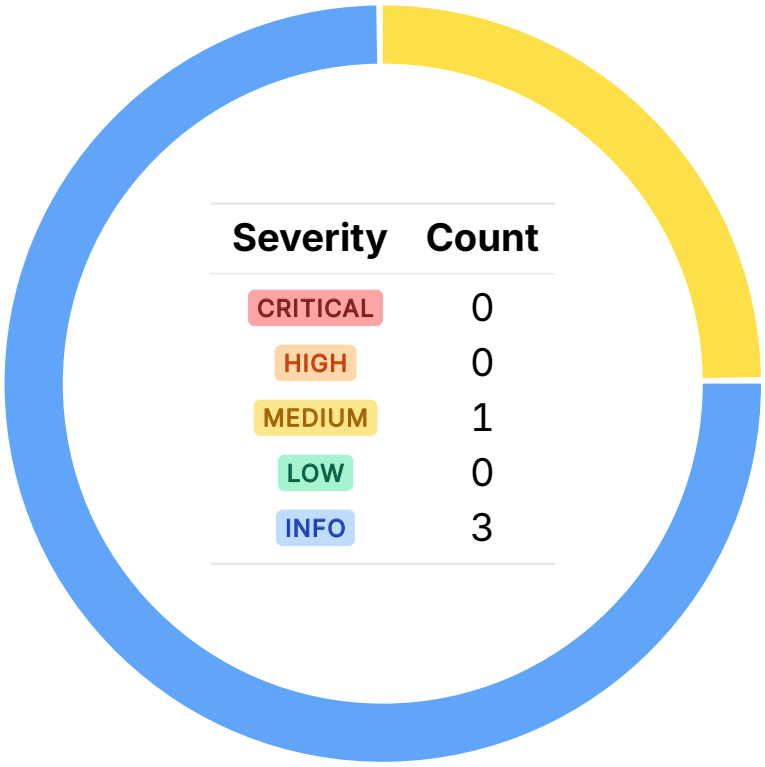
A brief description of the program is as follows:

Name	Description
sol-cc	It sets up a connector to the Vault program deployed on Solana, built on the LayerZero OApp framework with upgradeable settings.
solana-vault	It sets up Orderly's Vault on the Solana, built on the LayerZero OApp/OFT codebase within the Anchor framework. The Solana Vault is connected to the Orderly chain via the LayerZero protocol.

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	1
LOW	0
INFO	3

03 — 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-OLN-ADV-00	MEDIUM	RESOLVED	User can change the authority of his ATA which could cause DoS on an OApp with ordered delivery.

Denial Of Service Due To Authority Change MEDIUM OS-OLN-ADV-00

Description

In `OAppLzReceive` instruction, the `receiver_token_account` is used to send the tokens received through LayerZero. The `receiver_token_account` is the ATA for the `receiver` on `token_mint`.

While transferring the received funds, an event is emitted instead of erroring out if the `receiver_token_account` is frozen, presumably to ensure the user is not able to fail the execution of the instruction by freezing the token account.

>_ src/instructions/oapp_instr/oapp_lz_receive.rs RUST

```rust
if ctx.accounts.receiver_token_account.is_frozen() {
    emit!(Into::<FrozenWithdrawn>::into(vault_withdraw_params.clone()));
} else {
    transfer(
        ctx.accounts
            .transfer_token_ctx()
            .with_signer(&[&vault_authority_seeds[..]]),
        amount_to_transfer,         //  should be u64 here
    )?;
    emit!(Into::<VaultWithdrawn>::into(vault_withdraw_params.clone()));
}
```

Although, the user could change the authority of the `receiver_token_account` to a different public key other than `receiver` to make the instruction fail.

Remediation

Implement token tracking and a withdrawal function for the admin.

Patch

Resolved in 09c4980.

04 — 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-OLN-SUG-00	The admin transfer process may be enhanced for improved safety and functionality.
OS-OLN-SUG-01	Allowing updates to the `orderDelivery` setting after deployment risks losing unexecuted messages when switching from unordered to ordered execution.
OS-OLN-SUG-02	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.

Enhancing Admin Transfer Process OS-OLN-SUG-00

Description

The current design in `transferAdmin` directly updates the `admin` field in a single step, there is no confirmation step. Once the transaction is submitted, the admin authority change is irreversible. This may result in a denial of service if the current admin accidentally sends an unintended input as a parameter while executing an admin change.

>_ *instructions/oapp_instr/transfer_admin.rs* RUST

```rust
impl TransferAdmin<'_> {
    pub fn apply(ctx: &mut Context<TransferAdmin>, params: &TransferAdminParams) -> Result<()> {
        ctx.accounts.oapp_config.admin = params.admin;
        // TODO:call endpoint to update delegate
        Ok(())
    }
}
```

Furthermore, the `TODO` comment suggests calling an external endpoint to update the delegate associated with the admin on the LayerZero endpoint program. However, the admin may not always serve as the delegate because the delegate may be updated independently utilizing the `SetDelegate` instruction.

Remediation

Implement a two-step process for admin change, such that the new admin address is set in the first step to initiate the transfer, and the new admin consequently confirms the transfer in the second step. Since the admin can update the delegate later, updating the delegate here may not be necessary.

Avoid Loss of Unexecuted Messages OS-OLN-SUG-01

Description

Restrict updates to the `orderDelivery` flag after deployment to ensure that unexecuted messages are not lost when switching from unordered to ordered execution. To prevent sending transactions while the contract is in the process of transitioning the execution model, utilize `pause` from OpenZeppelin's `Pausable` contract, to freeze the sender contract.

Remediation

Incorporate the pausing functionality to freeze the contract.

Code Maturity OS-OLN-SUG-02

Description

1. Within `SetVault` instruction, when the signer is not the admin the `VaultError::InvalidVaultOwner` error is utilized instead of `OAppError::Unauthorized` when the `oapp_config` account's admin field does not match the admin signer. Utilize the `OAppError::Unauthorized` error in this case for consistency and to avoid misunderstandings about the cause of errors.

>_ instructions/vault_instr/set_vault.rs RUST

```rust
pub struct SetVault<'info> {
    #[account(mut)]
    pub admin: Signer<'info>,
    [...]
    #[account(
        seeds = [OAPP_SEED],
        bump = oapp_config.bump,
        has_one = admin @ VaultError::InvalidVaultOwner,
    )]
    pub oapp_config: Account<'info, OAppConfig>,

    pub system_program: Program<'info, System>,
}
```

2. `SetToken::apply` assigns `params.mint_account` to `allowed_token.mint_account` but utilizes `ctx.accounts.mint_account.decimals` to set `allowed_token.token_decimals`, creating a potential inconsistency where the `allowed_token.mint_account` does not match the `mint` account from which the decimals are derived. Utilize `ctx.accounts.mint_account` instead of `params.mint_account` when setting `allowed_token.mint_account`.

>_ instructions/vault_instr/set_token.rs RUST

```rust
pub fn apply(ctx: &mut Context<SetToken>, params: &SetTokenParams) -> Result<()> {
    ctx.accounts.allowed_token.mint_account = params.mint_account;
    ctx.accounts.allowed_token.token_hash = params.token_hash;
    ctx.accounts.allowed_token.token_decimals = ctx.accounts.mint_account.decimals;
    ctx.accounts.allowed_token.allowed = params.allowed;
    [...]
}
```

3. To enhance safety, it would be appropriate to add a functions to withdraw excess fees from the contract. This addresses the current flow where fees are topped up to the contract instead of paying directly from the ledger, ensuring any excess funds may be removed and do not remain trapped without a withdrawal mechanism.
4. Implement admin functions to close accounts, such as `AllowedToken` and `AllowedBroker`, to reclaim lamports from un-utilized accounts.

Remediation

Implement the above-mentioned suggestions.

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.