SHERLOCK

# Sherlock Security Review For Orderly Network

| | |
|---|---|
| Public contest prepared for: | **Orderly Network** |
| Lead Security Expert: | **g** |
| Date Audited: | **October 15 - October 27, 2024** |

## Introduction

Allow users to deposit and withdraw their USDC to and from the contract via the Solana chain.

## Scope

Repository: OrderlyNetwork/sol-cc

Branch: add-ut-using-foundry

Audited Commit: dc99b068cda9a6067b35edf629acd1730e5982a3

Final Commit: d0bbdda7ec3b9cdbcc2829d9852b9e86eac67c36

---

Repository: OrderlyNetwork/solana-vault

Branch: dev

Audited Commit: bd8b6dbeb3300319fd9dad262298ec0cd1152344

Final Commit: ee16d85e0be72e0527ad8e630fd98b1dd0de52f5

---

For the detailed scope, see the contest details.

## Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

## Issues found

| Medium | High |
|---|---|
| 1 | 2 |

## Issues not fixed or acknowledged

| Medium | High |
|---|---|
| 0 | 0 |

## Security experts who found valid issues

| | | |
|---|---|---|
| g | 0rpse | Silvermist |
| krikolkk | ubermensch | infect3d |
| LZ_security | chinepun | 0xBoboShanti |
| 0xNirix | dod4ufn | sh1v |
| Q7 | S3v3ru5 | pashap9990 |
| shaflow01 | Tendency | xKeywordx |

# Issue H-1: [H-1]

Source: https://github.com/sherlock-audit/2024-09-orderly-network-solana-contract-judging/issues/40

## Found by

0rpse, 0xNirix, LZ_security, Q7, S3v3ru5, Tendency, chinepun, dod4ufn, g, krikolkk, pashap9990, sh1v, shaflow01, ubermensch, xKeywordx

## Summary

There are no checks to ensure that the `deposit_token` matches the `allowed_token` in the `solana_vault::deposit` function. This allows an attacker to deposit any tokens and get minted USDC on the other chain. The system assumes that an allowed token is deposited every time and this assumption is wrong.

## Root Cause

In deposit.rs, the `Deposit` struct lacks a constraint to verify that `deposit_token.key()` matches the `allowed_token.mint_account`.

This check could also be added inside the `solana_vault::deposit` or the `deposit.rs::apply` functions directly, but currently, none of these functions have this check either.

`deposit_params.token_hash` is a user-controlled input param provided by the user when they call the `solana_vault::deposit` function. It is intended to represent the hash of the token the user is depositing. The problem is that there is no enforced relationship with `deposit_token`. There is no constraint that enforces that `deposit_params.token_hash` corresponds to the actual `deposit_token` mint provided in the accounts.

## Internal pre-conditions

1. The `allowed_token` account is configured to accept a specific token (USDC) with `allowed_token.allowed==true`.
2. The `allowed_token.mint_account` is set to the mint address of the allowed token (USDC).
3. The vault is initialized and operational, expecting deposits of the allowed token.

## External pre-conditions

1. The attacker possesses another token, e.g., WIF or another memecoin.

2. The attacker has a token account associated with this memecoin and owns, let's say 10.000 tokens which are worth 1.Assumeaprice/coinof0.0001.
3. The attacker knows the `token_hash` corresponding to the allowed token (USDC).

## Attack Path

The attacker calls the deposit Function and provides:

- `WIF` as `deposit_token`
- with an amount of `10.000`
- `deposit_params.token_hash` that corresponds to USDC token_hash
- includes the `allowed_token` that the Vault is configured to accept (USDC).

Program Execution:

- The function `deposit.rs::transfer_token_ctx` transfers the tokens from the attacker's `user_token_account` to the vault's `vault_token_account`.
- Due to the missing constraint, the program does not verify that `deposit_token.key()` matches `allowed_token.mint_account`.
- The `deposit.rs::apply` function increments `vault_authority.deposit_nonce` by 1.
- After that, the `apply` function creates the `VaultDepositParams` object, setting `token_hash` to be equal to whatever `token_hash` the user provided inside the `DepositParams` struct.
- The program records the deposit as if the allowed token (USDC) was deposited.
- It sends a message to the other chain indicating that the allowed token was deposited.

Resulting Impact:

- On the other chain, the attacker is credited with USDC equivalent to the amount of tokens deposited (10.000 USDC).
- The vault holds the WIF tokens, which are worth 1$ (for the sake of this example). The attacker can even create a random coin with no value and make the deposit.

## Impact

For the Protocol:

- The protocol suffers a financial loss equivalent to the amount of USDC incorrectly credited to the attacker on the other chain.
- The vault ends up holding unauthorized tokens instead of the intended allowed tokens.

For the Attacker:

- The attacker gains USDC on the other chain without depositing the equivalent value of the allowed token.
- The attacker can drain the vault.

## PoC

Not necessary. Check the `solana-vault/packages/solana/contracts/programs/solana-vault/src/instructions/vault_instr/deposit.rs::Deposit` struct definition here. There are no checks that enforce `deposit_token==allowed_token`.

You can also check the Solana Vault deposit function implementation `solana-vault/packages/solana/contracts/programs/solana-vault/src/lib.rs::deposit` here and see that it doesn't have this check, nor does the `solana-vault/packages/solana/contracts/programs/solana-vault/src/instructions/vault_instr/deposit.rs::apply` function -- check it here.

## Mitigation

Add a new `constraint` in `deposit.rs::Deposit` struct.

Update the `deposit_token` account definition in `deposit.rs` to include a `constraint` that ensures the `deposit_token` mint matches the `allowed_token.mint_account`.

```
#[account(
    constraint = deposit_token.key() == allowed_token.mint_account @
 ↪  VaultError::InvalidDepositToken
)]
```

## Discussion

**sherlock-admin2**

The protocol team fixed this issue in the following PRs/commits:
https://github.com/OrderlyNetwork/solana-vault/pull/4

**gjaldon**

This issue is fixed here. The recommended constraint was added to `deposit.rs::Deposit` struct.

## Issue H-2: A malicious user can withdrawals another user's money

Source: https://github.com/sherlock-audit/2024-09-orderly-network-solana-contract-judging/issues/99

### Found by

0rpse, 0xBoboShanti, Q7, S3v3ru5, Silvermist, Tendency, chinepun, dod4ufn, g, infect3d, krikolkk, shaflow01, ubermensch

### Summary

A shared vault authority signing mechanism will cause unauthorized withdrawals for users, as User A can withdraw funds belonging to User B.

### Root Cause

In the OAppLzReceive, the `vault_authority_seeds` are shared across all users, allowing any user with valid withdrawal parameters to use the same PDA signing authority. As a result, any valid withdrawal request can be signed by the vault without distinguishing which user is performing the withdrawal.

Also, there is no check that the wallet receiving the funds belongs to the same user for whom the withdrawal request was initiated. The system only checks that the withdrawal message comes from a valid sender (peer.address == params.sender), but does not verify that the user account corresponds to the sender.

### Internal pre-conditions

*No response*

### External pre-conditions

*No response*

### Attack Path

1. User B initiates a withdrawal on the Ethereum side and a valid withdraw message is sent to the Solana.

2. User A accesses the valid withdrawal messages corresponding to User B's account and calls the function before User B.
3. Since there is no check to ensure the User A uses a withdrawal message corresponding to his account, the withdraw is successfully executed and User A steals User B's money.

## Impact

The attacker steals the entire withdrawn amount from User B's account without any corresponding loss.

## PoC

*No response*

## Mitigation

*No response*

## Discussion

**sherlock-admin2**

The protocol team fixed this issue in the following PRs/commits: https://github.com/OrderlyNetwork/solana-vault/pull/4/commits/a9f56db5e63562df9eb6a39803f3df12b7959032

**gjaldon**

A check was added to `oapp_lz_receive()` that validates that the recipient of the withdrawal is the `receiver` specified in the withdrawal payload.

## Issue M-1: Missing LayerZero Ordered Execution Option For Orderly Chain Messages

Source: https://github.com/sherlock-audit/2024-09-orderly-network-solana-contract-judging/issues/146

The protocol has acknowledged this issue.

### Found by

0xNirix, LZ_security, g, krikolkk

### Summary

Missing ordered execution enforcement will cause message rejection vulnerability for users as unordered messages will update balances on Orderly Chain's ledger but fail on Solana vault due to message ordering mismatch.

### Root Cause

In https://github.com/sherlock-audit/2024-09-orderly-network-solana-contract/blob/main/sol-cc/contracts/SolConnector.sol#L92 the withdraw implementation only applies basic gas/value options without ordered execution option: solidityCopybytes memory withdrawOptions = OptionsBuilder.newOptions().addExecutorLzReceiveOption( msgOptions[uint8(MsgCodec.MsgType.Withdraw)].gas, msgOptions[uint8(MsgCodec.MsgType.Withdraw)].value ); LayerZero will not guarantee ordered message delivery in absence of this option.

### Internal pre-conditions

1. Ordered delivery is enabled on Solana vault side and Orderly chain
2. Multiple withdrawals are processed in quick succession

### External pre-conditions

*No response*

### Attack Path

1. User A and User B submit withdrawals close together
2. SolConnector sends messages with only gas/value options configured

3. LayerZero messages arrive out of order at Solana vault due to missing ordered execution option
4. Solana vault rejects out-of-order messages with InvalidInboundNonce error
5. Ledger contract has already updated balances on Orderly Chain
6. Balance state becomes inconsistent between chains

## Impact

Users suffer from state inconsistency where their balances are reduced on Orderly Chain's ledger but funds remain locked in Solana vault due to message rejection. User may effectively lose funds and to resolve this extensive manual intervention may be required by admin.

## PoC

*No response*

## Mitigation

*No response*

## Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.