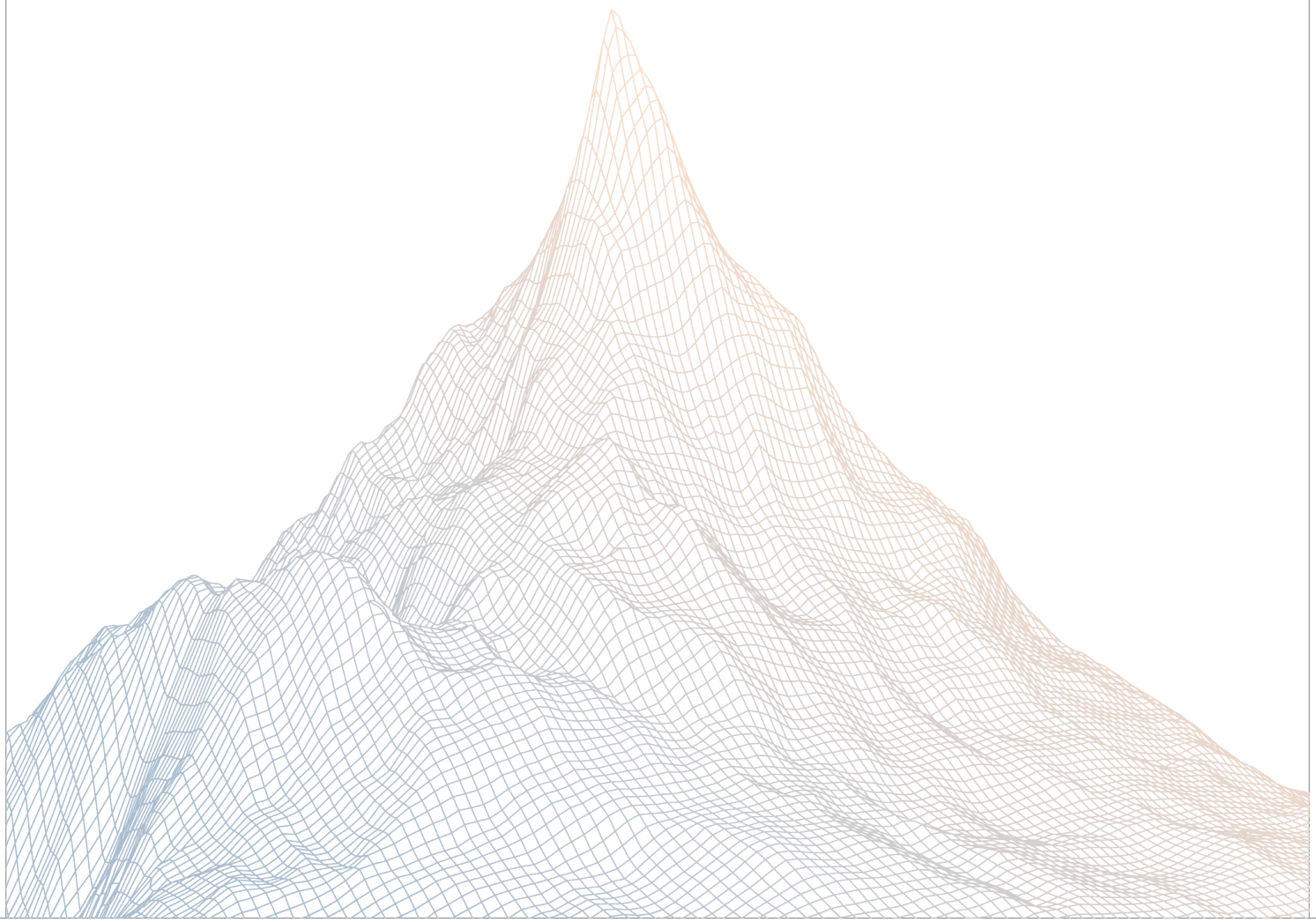
Zenith

Orderly

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: March 17th to March 25th, 2025

AUDITED BY: J4X
Peakbolt

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Orderly

Orderly is a combination of an orderbook-based trading infrastructure and a robust liquidity layer offering perpetual futures orderbooks. Unlike traditional platforms, Orderly doesn't have a front end; instead, it operates at the core of the ecosystem, providing essential services to projects built on top of it.

Our DEX white-label solution is carefully crafted to save builders time and capital while granting access to our bootstrapped liquidity.

2.2 Scope

The engagement involved a review of the following targets:

Target	omnichain-ledger
Repository	https://github.com/OrderlyNetwork/omnichain-ledger
Commit Hash	`6221bd1bee53f3de370176d4757827d9262f6764`
Files	`Diff in OrderlyNetwork/omnichain-ledger/compare/main ... audit-for-solana-2`

Target	solana-proxy
Repository	https://github.com/OrderlyNetwork/solana-proxy
Commit Hash	`d38763dacaa5dc15b0b142faa6c2cf30aa119324`

Files	`instructions/set_backward_fee.rs`
	`instructions/set_peer_config.rs`
	`instructions/send_request.rs`
	`instructions/init_proxy.rs`
	`instructions/lz_receive_types.rs`
	`instructions/quote_claim.rs`
	`instructions/transfer_admin.rs`
	`instructions/set_delegate.rs`
	`instructions/msg_codec.rs`
	`instructions/set_pause.rs`
	`instructions/mod.rs`
	`instructions/send_claim.rs`
	`instructions/set_accounts_list.rs`
	`instructions/lz_receive.rs`
	`instructions/quote_request.rs`
	`instructions/withdraw_fee.rs`
	`events.rs`
	`lib.rs`
	`state/peer_config.rs`
	`state/proxy_config.rs`
	`state/backward_fee.rs`
	`state/claim_data.rs`
	`state/mod.rs`
	`state/seeds.rs`
	`errors.rs`

2.3 Audit Timeline

March 17, 2025	Audit start
March 25, 2025	Audit end
March 25, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	6
Informational	2
Total Issues	**8**

3

Findings Summary

ID	Description	Status
L-1	Failed messages in LedgerOApp can be retried and executed by anyone	Acknowledged
L-2	Missing whenNotPaused modifier for lzCompose()	Resolved
L-3	UnstakeNow will take less than 5% due to rounding	Acknowledged
L-4	Backwards fee will be unnecessarily charged in case of $ESORDER reward claim	Acknowledged
L-5	Unable to cancel claim and retrieve rent for ClaimData when paused	Resolved
L-6	Tokens sent using OFT could get stuck	Resolved
I-1	Consider removing unused code in solana-proxy program	Acknowledged
I-2	Storage gap is incorrectly adjusted for LedgerOCCManager	Acknowledged

4

Findings

4.1 Low Risk

A total of 6 low risk findings were identified.

[L-1] Failed messages in `LedgerOApp` can be retried and executed by anyone

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- LedgerOApp.sol#L74-L100

Description:

`LedgerOApp._lzReceive()` is used to receive Oapp message that was sent from the Solana Proxy and pass it to `LedgerOCCManager` via `ledgerOappReceive()`. The `LedgerOApp` is configured for unordered delivery (default setting) but that is fine as long as the LZ Endpoint and Executor will process them in a first-come-first-serve manner.

However, when LedgerOApp is paused, `_lzReceive()` will revert and the received Oapp messages will sit in the LZ Endpoint, which can then be retried and executed via `Endpoint.lzReceive()` when `LedgerOApp` is unpaused. That will then trigger `LedgerOapp.lzReceive()`.

The issue is that by default anyone can call `Endpoint.lzReceive()` as indicated in the docs below.

> Once the message has fulfilled the OApp's configured Security Stack, any caller (e.g., Executor) can commit the verified packet nonce to the destination Endpoint and execute the corresponding message via `lzReceive` for the receiving application.

That will allow anyone to be able to execute the pending Oapp messages, in any order, such that it is no longer first-come-first-serve. This will affect the users as it will not function as expected.

Example Scenario

1. User A sends request for `withdrawOrder` and then another request for `createOrderUnstakeRequest`.
2. LedgerOApp is paused, and both request are not executed and now sit in LZ Endpoint pending retry.
3. LedgerOApp is unpaused. Attacker calls `Endpoint.lzReceive()` to execute `createOrderUnstakeRequest` first. That will cause the unlockTimestamp for withdrawable Order token to be reset again. Now `withdrawOrder` request will fail, and User A has to wait till the lock period is over.

```
function _lzReceive(
    Origin calldata _origin,
    bytes32 /*_guid*/,
    bytes calldata _message,
    address /*_executor*/,
    bytes calldata /*_extraData*/
) internal override whenNotPaused {  // add whenNotPaused modifier
    uint32 solanaEid = ILedgerOCCManager(occManagerAddr).solanaEid();
    uint256 solanaChainId
    = ILedgerOCCManager(occManagerAddr).eid2ChainId(solanaEid);
    if (_origin.srcEid == solanaEid) {
        SolanaVaultMessage memory solanaVaultMessage
    = _message.decodeSolanaVaultMessage();
        require(solanaVaultMessage.payloadType.checkVaultPayloadType(),
    "LedgerOApp: invalid vault payload type");
        require(solanaChainId != 0, "LedgerOApp: Solana chain id not set");
        OCCVaultMessage memory occVaultMessage = OCCVaultMessage({
            chainedEventId: 0, // @dev: chainEventId will be updated in
    OCCManager for solana proxy
            srcChainId: solanaChainId,
            token: solanaVaultMessage.token,
            tokenAmount: uint256(0),
            sender: solanaVaultMessage.sender,
            payloadType: solanaVaultMessage.payloadType,
            payload: solanaVaultMessage.payload
        });

    ILedgerOCCManager(occManagerAddr).ledgerOappReceive(occVaultMessage);
    } else {
        revert("LedgerOApp: only Solana chain supported");
    }
}
```

Recommendations:

In `_lzReceive()`, whitelist the `_executor` to LZ Executor, the message sender and the protocol administrator.

Orderly: Acknowledged

[L-2] Missing `whenNotPaused` modifier for `lzCompose()`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- ProxyLedger.sol#L219-L225

Description:

The Proxy contracts are designed to stop any sending/receiving of LZ messages when paused is enabled. This is the case for Solana Proxy program, where sending/receive functions check for the pause flag.

However, `ProxyLedger` does not fully align to this design as it lacks a `whenNotPaused` modifier for `lzCompose()`. This allows any inflight messages that were sent before paused, to be processed, deviating from the Solana Proxy design.

```
function lzCompose(
    address from,
    bytes32 /*guid*/,
    bytes calldata _message,
    address /*executor*/,
    bytes calldata /*_extraData*/
) external payable {
```

Recommendations:

Add `whenNotPaused` modifier to `lzCompose()`.

Orderly: Fixed in @1dac50c8140...

Zenith: Verified.

[L-3] UnstakeNow will take less than 5% due to rounding

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- omnichain-ledger/contracts/lib/Staking.sol#L190-L191

Description:

The `_unstakeOrderNow()` function is used so that users can instantly unstake their funds with a penalty. 5% of the unstaked amount will be kept by the protocol. This amount is calculated in the following snippet:

```
orderAmountForCollect = (_amount * UNSTAKE_NOW_COLLECT_PERCENT) / 100; // 5%
    of withdrawn amount
orderAmountForWithdraw = _amount - orderAmountForCollect;
```

As `orderAmountForCollect` is the amount that will be kept be the protocol and rounded down, it will be less than 5% most of the time. As no minimum is enforced, a user could (theoretically if there was no gas cost involved) unstake in batches of 19 tokens and bypass the restriction altogether.

Recommendations:

We recommend ceiling the division.

Orderly: Acknowledged

[L-4] Backwards fee will be unnecessarily charged in case of $ESORDER reward claim

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- solana-proxy/programs/solana-proxy/src/instructions/send_claim.rs#L125

Description:

The cross-chain implementation implements a backward fee. This fee will be charged to users additionally if their call requires the protocol to do a cross-chain callback. The fee is intended to cover this additional cost. When a user sends a claim via the `send_claim` instruction, he will always be charged the backward fee.

```
let backward_fee = ctx.accounts.backward_fee.order_backward_fee;

if backward_fee > 0 {
    program::invoke(
        &system_instruction::transfer(ctx.accounts.user.key,
    &ctx.accounts.proxy_config.key(), backward_fee),
        &[ctx.accounts.user.to_account_info(),
    ctx.accounts.proxy_config.to_account_info()],
    )?;
}
```

However, when looking at the implementation on the EVM side, a backward message will only be sent if $ORDER rewards are claimed and not $ESORDER rewards.

```
if (claimedAmount != 0) {
    if (token == LedgerToken.ESORDER) {
        _stake(_user, _chainedEventId, _srcChainId, token, claimedAmount);
    } else if (token == LedgerToken.ORDER) {
        EvmLedgerMessage memory message = EvmLedgerMessage({
            dstChainId: _srcChainId,
            token: LedgerToken.ORDER,
            tokenAmount: claimedAmount,
            receiver: _user,
```

```
            payloadType: uint8(PayloadDataType.ClaimRewardBackward),
            payload: "0x0"
        });
        ILedgerOCCManager(occAdaptor).ledgerSendToVault(message);
    }
}
```

As a result, the user will be charged a backward fee without an actual backward message.

Recommendations:

We recommend allowing the user to pass the token he wants to claim rewards on the Solana side. Then, the backward fee will only be charged based on if it is $ESORDER or $ORDER. On the EVM side, the code can verify that the token provided by the user matches the token used in the distribution.

This also prevents abuse, as if a user sets $ESORDER to claim $ORDER, his claim would just be denied on the EVM side.

Orderly: Acknowledged

[L-5] Unable to cancel claim and retrieve rent for `ClaimData` when paused

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- send_claim.rs

Description:

Users are required to init the `ClaimData` account via `submit_proof()` first, in order to proceed with `send_claim()`. The `ClaimData` account will then be closed in `send_claim()`, where the rent is refunded to the requesting user.

However, if the solana proxy program is paused right after a `submit_proof()`, the user will not able to proceed with `send_claim()`. That means there is no other mechanism to close `ClaimData` account and retrieve the rent till the protocol is unpaused.

Recommendations:

Consider allowing the user to cancel the claim request and close the `ClaimData` when the protocol is paused.

Orderly: Fixed in @8ec05514738...

Zenith: Verified. Resolved with a `cancel_claim` instruction.

[L-6] Tokens sent using OFT could get stuck

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- src/instructions/send.rs#L52

Description:

The EVM side implementation of the OFT will revert if the compose message is anything other than

```
require(
    PayloadDataType(occVaultMessage.payloadType) == PayloadDataType.Stake,
    "LedgerOCCManager: Only Stake payload is supported through Solana OFT
    channel"
```

However, this is not restricted on the Solana side. As a result, a user could accidentally send tokens with a message that could not be used on the other side and would get stuck.

Recommendations:

We recommend checking if the message is the `Staking` message and reverting otherwise.

Orderly: Resolved with @291cb69520..., @cf5cf45d7... & @e2c84430db...

Zenith: Verified

4.2 Informational

A total of 2 informational findings were identified.

[I-1] Consider removing unused code in solana-proxy program

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- programs/solana-proxy/src/instructions/set_peer_config.rs#L47-L52

Description:

The Solana Proxy program has the following un-used code that can be removed for better maintainability and readability.

1. `solana_proxy::set_peer_config()` has unused code that allows setting outbound/inbound rate limiting but are not enforced.

```
impl SetPeerConfig<'_> {
    pub fn apply(ctx: &mut Context<SetPeerConfig>, params:
    &SetPeerConfigParams) -> Result<()> {
        match params.config.clone() {
            ..
            PeerConfigParam::OutboundRateLimit(rate_limit_params) => {
                Self::update_rate_limiter(&mut
    ctx.accounts.peer_config.outbound_rate_limiter, &rate_limit_params)?;
            },
            PeerConfigParam::InboundRateLimit(rate_limit_params) => {
                Self::update_rate_limiter(&mut
    ctx.accounts.peer_config.inbound_rate_limiter, &rate_limit_params)?;
            },
        }
```

Recommendations:

1. Remove `PeerConfigParam::OutboundRateLimit` and `PeerConfigParam::InboundRateLimit`.

Orderly: Fixed in @9634169c3a...

Zenith: Verified. Resolved by removing unused code.

[I-2] Storage gap is incorrectly adjusted for `LedgerOCCManager`

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- LedgerOCCManager.sol#L390-L391

Description:

`LedgerOCCManager` has been updated with 5 new storage variables, along with the reduction in the storage gaps from 50 to 45.

However, the number of slots used by the new storage variables is actually 4, as both `uint32 public solanaEid` and `address public ledgerOappAddr;` will be packed into the same slot as they occupy less than 32 bytes, as shown below.

That means the storage gaps should be adjusted to 46 instead of 45.

```
contract LedgerOCCManager is Initializable, LedgerAccessControl,
    OCCAdapterDatalayout, UUPSUpgradeable {

    //@audit slot 0
    address public ledgerAddr;

    //@audit slot 1
    mapping(uint256 ⇒ address) public chainId2ProxyLedgerAddr;

    //@audit slot 2
    address public orderCollector;

    //@audit slot 3
    mapping(bytes32 ⇒ address) public userSolana2EvmAddress;

    //@audit slot 4
    mapping(address ⇒ bytes32) public userEvm2SolanaAddress;

    //@audit slot 5
    uint32 public solanaEid;
```

```
    //@audit slot 5
    address public ledgerOappAddr;

    //@audit slot 6
    uint256 public solanaChainEventId;

    ..
    /// gap for upgradeable
    uint256[45] private __gap;
```

Recommendations:

```
    /// gap for upgradeable
    uint256[45] private __gap;
    uint256[46] private __gap;
```

Zenith: Fixed in @52da3f6a520...

Zenith: Verified.