Wormhole Solana

Security Assessment

June 25th, 2024 — Prepared by OtterSec

Ajay Shankar Kunapareddy d1r3wolf@osec.io

Robert Chen r@osec.io

Table of Contents

Executive Summary	**2**
Overview	2
Key Findings	2
Scope	**3**
Findings	**4**
Vulnerabilities	**5**
OS-ELL-ADV-00 | Custody Token Account Closing DoS	6
OS-ELL-ADV-01 | Empty Token Account DoS	8
OS-ELL-ADV-02 | Missing Endpoint Check	9
OS-ELL-ADV-03 | Unchecked Refund Token	10
OS-ELL-ADV-04 | Inaccurate Max Account Size Calculation	11
OS-ELL-ADV-05 | Mismatched Chain ID	12
OS-ELL-ADV-06 | Inaccurate Storage Size Calculation	13
General Findings	**14**
OS-ELL-SUG-00 | Missing Validations	15
OS-ELL-SUG-01 | Code Maturity	16
OS-ELL-SUG-02 | Removal Of Dead Code	17
OS-ELL-SUG-03 | Closed Initial Offer Token Account	18

Appendices

Vulnerability Rating Scale	**19**
Procedure	**20**

01 — Executive Summary

Overview

Wormhole Foundation engaged OtterSec to assess the `example-liquidity-layer` program. This assessment was conducted between May 23rd and June 15th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 11 findings throughout this audit engagement.

In particular, we identified several denial of service vulnerabilities, including the assumption during settlement that the transferred amount matches the deposit. An attacker may send a smaller amount, failing the settlement and preventing other auctions from closing their tokens (OS-ELL-ADV-00). A similar issue allowed for bypassing the current account existence checks in the auction system by closing the accounts before settlement and reopening them with different tokens (OS-ELL-ADV-01).

Furthermore, the preparation of a market order does not include a refund token in its hash calculation, allowing it to be manipulated (OS-ELL-ADV-03). The settlement process skips checking if the order originated locally, which enables the possibility of a remote order's endpoint appearing as local and delivering the token on Solana instead of the intended chain (OS-ELL-ADV-02).

We also made recommendations to include additional validations within the codebase for improved security (OS-ELL-SUG-00) and suggested the need to ensure adherence to coding best practices (OS-ELL-SUG-01). Additionally, we advised the removal of unused code for increased readability and maintainability (OS-ELL-SUG-02).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/wormhole-foundation/example-iquidity-layer. This audit was performed against commit 78d6f22.

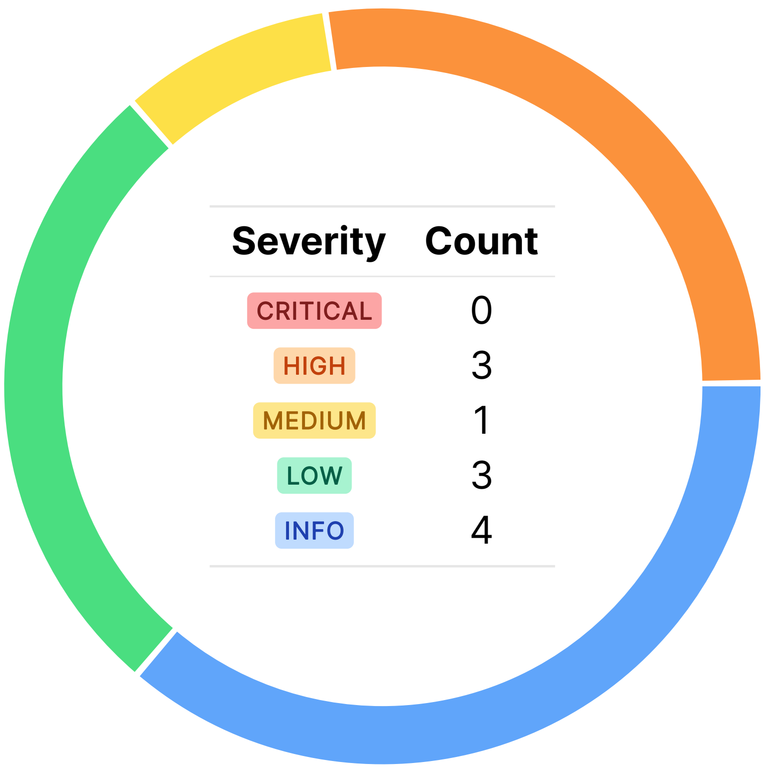
A brief description of the programs is as follows:

Name	Description
example–liquid-ity–layer	This program utilizes the Wormhole Circle Integration contract to facilitate cross-chain transfers of USDC (along with arbitrary messages) to custom smart contracts on any CCTP-enabled blockchain.

03 — Findings

Overall, we reported 11 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	3
MEDIUM	1
LOW	3
INFO	4

04 — 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-ELL-ADV-00	HIGH	RESOLVED	Settlement assumes the amount in the `prepared_custody_token` account matches the `CCTP` deposit (`order.amount_in`). An attacker may send a smaller amount, causing settlement to fail since closing the `prepared_custody_token` account fails with non-zero funds.
OS-ELL-ADV-01	HIGH	RESOLVED	The current token account existence checks in the auction system can be bypassed by closing the accounts and reopening them with a different mint. This results in a denial of service during token transfers.
OS-ELL-ADV-02	HIGH	RESOLVED	The `settle_auction_none_local` process skips checking whether the order originated locally, which enables the possibility of a remote order's endpoint appearing as local and delivering the token on Solana instead of the intended chain.
OS-ELL-ADV-03	MEDIUM	RESOLVED	`prepare_market_order` does not include `refund_token` in its hash calculation, allowing attackers to exploit market orders by setting a bad `refund_token`.
OS-ELL-ADV-04	LOW	RESOLVED	Within `auction_history`, the maximum number of entries (`MAX_ENTRIES`) in an auction history account is miscalculated.
OS-ELL-ADV-05	LOW	RESOLVED	`update_cctp_router_endpoint` does not verify if the provided chain ID matches the router endpoint's chain failing seed checks.
OS-ELL-ADV-06	LOW	RESOLVED	`compute_size` miscalculates storage size for `PreparedFill` accounts on Solana, assuming eight bytes for a `u8` field.

Custody Token Account Closing DoS HIGH OS-ELL-ADV-00

Description

There is a potential vulnerability related to the settlement instructions for auctions involving the Wormhole `CCTP` bridge. The code assumes the total transferable amount equals `order.amount_in` retrieved from the `fastVAA`. However, during the settlement process, it does not verify if this amount actually matches the funds in the `prepared_custody_token` account.

>_ *auction/prepare_settlement/cctp.rs* rust

```rust
fn handle_prepare_order_response_cctp(
    ctx: Context<PrepareOrderResponseCctp>,
    args: CctpMessageArgs,
) -> Result<()> {
    [...]
    let fast_vaa = ctx.accounts.fast_order_path.fast_vaa.load_unchecked();
    let order = LiquidityLayerMessage::try_from(fast_vaa.payload())
        .unwrap()
        .to_fast_market_order_unchecked();

    let amount_in = order.amount_in();
    ctx.accounts
        .prepared_order_response
        .set_inner(PreparedOrderResponse {
            bump: ctx.bumps.prepared_order_response,
            info: PreparedOrderResponseInfo {
               [...]
                amount_in,
                sender: order.sender(),
                redeemer: order.redeemer(),
                init_auction_fee: order.init_auction_fee(),
            },
            to_endpoint: ctx.accounts.fast_order_path.to_endpoint.info,
            redeemer_message: order.message_to_vec(),
        });
    [...]
}
```

Before settling the `PrepareOrderResponse`, if a small amount of tokens is transferred to the `prepared_custody_token` account, when the settlement instruction tries to close the `prepared_custody_token` account via `token::close_account`, it fails with non-zero funds because only the `order.amount_in` is transferred from that account instead of the total balance.

Remediation

Instead of relying solely on `order.amount_in` from the fast `VAA`, the code should use the actual token balance in the `prepared_custody_token` account before attempting to close it.

Patch

Resolved in 307cc28.

Empty Token Account DoS HIGH OS-ELL-ADV-01

Description

Within the current implementation of `prepare_order_execution` and `improve_offer` functions, the `data_is_empty` checks have a potential vulnerability that could result in denial-of-service issues. The code relies on `data_is_empty()` function to determine if the respective token account exists.

>_ *processor/auction/execute_fast_order/mod.rs* rust

```rust
fn prepare_order_execution(accounts: PrepareFastExecution) -> Result<PreparedOrderExecution> {
    [...]
        // If the initial offer token account doesn't exist anymore, we have nowhere to send the
        // init auction fee. The executor will get these funds instead.
        if !initial_offer_token.data_is_empty() {
            if best_offer_token.key() != initial_offer_token.key() {
                // Pay the auction initiator their fee.
                [...]
                    init_auction_fee,
                )?;
                // Because the initial offer token was paid this fee, we account for it here.
                remaining_custodied_amount =
                    remaining_custodied_amount.saturating_sub(init_auction_fee);
            }
            [...]
        }
    [...]
}
```

This vulnerability can be exploited by placing an initial offer or best offer, then closing the respective token accounts and reopening them with a different mint before the auction is supposed to execute. Since there is new token data in those accounts, the `data_is_empty` checks would pass, allowing the protocol to proceed with the token transfer, which will fail due to the incorrect mint, creating a DoS for auction execution.

Remediation

Implement checks to ensure the account is a token account and verify the mint for the token accounts instead of relying on `data_is_empty`.

Patch

Resolved in #188 and 6ab5d8e.

Missing Endpoint Check HIGH OS-ELL-ADV-02

Description

`settle_auction_none_local` fails to verify the endpoint information within the `PreparedOrderResponse` account. It does not check if the `to_endpoint` field in the `PreparedOrderResponse` account is set to `Local` before proceeding with settlement on the Solana chain. Consequently, this may result in the settlement process attempting to deliver orders from a remote chain on Solana instead of routing them to the target chain, resulting in lost or inaccessible funds for the legitimate recipient.

>_ *auction/settle/none/local.rs* rust

```rust
pub fn settle_auction_none_local(ctx: Context<SettleAuctionNoneLocal>) -> Result<()> {
    [...]
    let super::SettledNone {
        user_amount: amount,
        fill,
    } = super::settle_none_and_prepare_fill(
        super::SettleNoneAndPrepareFill {
            prepared_order_response: &mut ctx.accounts.prepared.order_response,
            prepared_custody_token,
            auction: &mut ctx.accounts.auction,
            fee_recipient_token: &ctx.accounts.fee_recipient_token,
            custodian,
            token_program,
        },
        ctx.bumps.auction,
    )?;
    [...]
}
```

Remediation

The `settle_auction_none_local` function should verify that `order_response.to_endpoint` is `Local` before proceeding with the settlement.

Patch

Resolved in #194.

Unchecked Refund Token MEDIUM OS-ELL-ADV-03

Description

`prepare_market_order` calculates a hash based on several order details utilized for generating a temporary program-derived address (PDA)(`transfer_authority`) for token transfer authorization. However, this hash calculation does not include the `refund_token` field (the user's designated token account for refunds). An attacker may exploit this by preparing an order on an already-utilized `prepared_order` account for which Wormhole or `CCTP` messages are already created, setting a random address as the `refund_token`.

>_ *token-router/src/processor/market_order/prepare.rs* rust

```rust
pub fn prepare_market_order(
    ctx: Context<PrepareMarketOrder>,
    args: PrepareMarketOrderArgs,
) -> Result<()> {
    let hashed_args = args.hash();

    let PrepareMarketOrderArgs {
        amount_in,
        min_amount_out,
        target_chain,
        redeemer,
        redeemer_message,
    } = args;
    [...]
}
```

When someone initiates the subsequent `place_cctp` instruction with this already used `prepared_order` account, `place_cctp` will fail because the `Wormhole` and `CCTP` messages are already created in the respective PDAs. Now, there is only one way for the funds in the `prepared_order` account to be accessed: through the `close_prepare_order` instruction, which will transfer the funds to the `refund_token` specified by the attacker.

Remediation

Include `refund_token` field in the hash calculation used for generating the `transfer_authority` PDA. This prevents tampering with the `refund_token` account utilized for the `prepared_order`.

Patch

Resolved in ed87b5b.

Inaccurate Max Account Size Calculation LOW OS-ELL-ADV-04

Description

In `machine_engine::AuctionHistory`, the code aims to calculate the maximum number of `AuctionEntry` objects that will fit within an `AuctionHistory` account. It takes the maximum account size as $10 * 1024 * 1000$. However, based on `MAX_PERMITTED_DATA_LENGTH` defined by Solana, it should be $10 * 1024 * 1024$. This underestimates the actual available space for storing auction entries, as the calculated `MAX_ENTRIES` will be lower than the actual maximum number of entries that would fit in the account.

>_ matching-engine/src/state/auction_history.rs rust

```rust
impl AuctionHistory {
    pub const SEED_PREFIX: &'static [u8] = b"auction-history";

    pub const START: usize = 8 + AuctionHistoryHeader::INIT_SPACE + 4;

    cfg_if::cfg_if! {
        if #[cfg(feature = "integration-test")] {
            pub const MAX_ENTRIES: u32 = 2;
        } else {
            #[allow(clippy::as_conversions)]
            #[allow(clippy::cast_possible_truncation)]
            #[allow(clippy::integer_division)]
            pub const MAX_ENTRIES: u32 = ((10 * 1024 * 1000 - Self::START) /
                ↪  AuctionEntry::INIT_SPACE) as u32;
        }
    }
}
```

Remediation

Update the calculation of `MAX_ENTRIES` to use the correct value for the maximum account size.

Patch

Resolved in #189.

Mismatched Chain ID LOW OS-ELL-ADV-05

Description

In `matching_engine::update_cctp_router_endpoint`, there is no proper validation to ensure that `args.chain` equals the existing chain ID `router_endpoint.chain` before invoking `handle_add_cctp_router_endpoint`, which updates the router endpoint with args.

>_ *matching-engine/src/composite/mod.rs* rust

```rust
#[derive(Accounts)]
pub struct ExistingMutRouterEndpoint<'info> {
    #[account(
        mut,
        seeds = [
            RouterEndpoint::SEED_PREFIX,
            &endpoint.chain.to_be_bytes()
        ],
        bump = endpoint.bump,
    )]
    pub endpoint: Account<'info, RouterEndpoint>,
}
```

Consequently, when any instruction attempts to access the `router_endpoint` account using the `ExistingMutRouterEndpoint` Accounts, the PDA address derivation will fail if `router_endpoint.chain` is incorrect since it was used as a seed. Allowing the `router_endpoint.chain` to update without proper validation might lead to a denial-of-service for many instructions.

Remediation

Ensure to explicitly check if `args.chain` matches `router_endpoint.chain` before proceeding in `update_cctp_router_endpoint`.

Patch

Resolved in #190.

Inaccurate Storage Size Calculation LOW OS-ELL-ADV-06

Description

In `PreparedFill::compute_size`, there is a potential inaccuracy in storage size calculation. The code assumes eight bytes of storage for one of the `u8` field. This is incorrect because a basic `u8` in Rust typically takes only one byte on the stack and in memory. `compute_size` may overestimate the actual storage required for a `PreparedFill` account, resulting in the allocation of extra lamports during account creation. This primarily affects cost estimation.

>_ *token-router/src/state/prepared_fill.rs* rust

```rust
impl PreparedFill {
    pub const SEED_PREFIX: &'static [u8] = b"fill";

    pub fn compute_size(payload_len: usize) -> usize {
        // We should not expect `payload_len` to cause this operation to overflow.
        #[allow(clippy::arithmetic_side_effects)]
        let out = 8 + 32 + 1 + 32 + 32 + FillType::INIT_SPACE + 8 + 2 + 32 + 4 + payload_len;

        out
    }
}
```

Remediation

Update `compute_size` to utilize the correct size for `u8` field, which is one byte.

Patch

Resolved in e7a9873.

05 — 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-ELL-SUG-00	Recommendations to include additional validations within the code base for improved security.
OS-ELL-SUG-01	Suggestions to modify the code base for improved clarity and readability.
OS-ELL-SUG-02	There are several instances of unused code that should be removed for increased readability and maintainability.
OS-ELL-SUG-03	The program does not check whether the initial offer token account is closed before utilizing it in `add_auction_history_entry`.

Missing Validations

OS-ELL-SUG-00

Description

Currently, the `redeemer_message` has no length restrictions. An unbounded message size may result in large storage costs or affect program execution. Incorporate a maximum length check (`MAX_LEN`) for the `redeemer_message` in both Solana and EVM token router programs.

For improved security, it will be beneficial to add an explicit address check for `token_router_program` in `LocalTokenRouter`.

>_ *matching-engine/src/composite/mod.rs* rust

```rust
#[derive(Accounts)]
pub struct LocalTokenRouter<'info> {
    /// CHECK: Must be an executable (the Token Router program), whose ID will be used to derive
        ↪ the
    /// emitter (router endpoint) address.
    #[account(executable)]
    pub token_router_program: UncheckedAccount<'info>,
    [...]
}
```

Remediation

Include the above validations into the code base.

Code Maturity

OS-ELL-SUG-01

Description

1. In the `PreparedOrderResponse` structure, within the `bump` field, store the `bump` value utilized to derive the program-derived address for the prepared token custody.
2. In `machine_engine::Initialize`, `paused_by` should be initially set to `owner`, based on the initialization in `token_router`.

Remediation

Implement the above-mentioned suggestions.

Removal Of Dead Code

OS-ELL-SUG-02

Description

The following instances of code may be removed from the system:

1. Remove the specified file: *common/src/admin/utils/upgrade.rs*.
2. The `upgrade_manager_program` account in the initialization instructions of `token_router` and `machine_engine` programs may be removed.
3. `destination_asset_info` field in `AuctionInfo` account may be removed.
4. `requireEmitter` and `requireEmitterLegacy` should be removed from `WormholeCctpTokenMessenger`.

Remediation

Ensure to remove the code mentioned in the above list.

Closed Initial Offer Token Account

OS-ELL-SUG-03

Description

There is a potential denial-of-service scenario in `machine_engine::add_auction_history_entry`. `add_auction_history_entry` attempts to add a new entry to the auction history for a settled auction.

>_ *matching-engine/src/processor/auction/history.rs* rust

```rust
pub struct AddAuctionHistoryEntry<'info> {
    ...

    #[account(
        mut,
        close = beneficiary,
        ...
    )]
    auction: Account<'info, Auction>,

    #[account(mut)]
    beneficiary: UncheckedAccount<'info>,

    #[account(
        token::authority = beneficiary,
        address = {
            match &auction.info {
                Some(info) => info.initial_offer_token,
                None => custodian.fee_recipient_token,
            }
        }
    )]
    beneficiary_token: Account<'info, token::TokenAccount>,
}
```

However, the code does not explicitly check if the `initial_offer_token` account (utilized when `AuctionInfo` is present) is still open and valid before attempting to utilize it. If an attacker manages to close the `initial_offer_token` account before `add_auction_history_entry` is called, it will result in a denial-of-service scenario as the transaction will fail because the `initial_offer_token` is closed.

Remediation

Perform a check to verify the state of the `initial_offer_token` account before utilizing it.

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 the 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.