ellipsis labs

Ellipsis Labs

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

William Wang defund@osec.io

Nicola Vella nick0ve@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Ellipsis Labs engaged OtterSec to perform an assessment of the `phoenix` program. This assessment of the source code was conducted between January 9th and February 8th, 2023. For more information on our auditing methodology, see Appendix C.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team to streamline patches and confirm remediation. We delivered final confirmation of the patches February 8th, 2023.

Key Findings

Over the course of this audit engagement, we produced 9 findings total. For a more detailed discussion of our analysis, see Discussion.

In particular, we found a Rust soundness issue in the core red-black tree implementation (OS-EPS-ADV-00). While this does not have immediate implications for the onchain orderbook, independent users of the library could experience undefined behavior.

We also noted a number of denial of service scenarios (OS-EPS-ADV-02, OS-EPS-ADV-03).

In addition, we provided recommendations around validating critical invariants (OS-EPS-SUG-00), optimizing data structures (OS-EPS-SUG-01), and general code quality to improve resilience.

Overall, we commend the Ellipsis Labs team for being responsive and knowledgeable. The codebase was well-written, documented, and tested prior to our audit, with clear attention to detail.

02 | Scope

A brief description of the programs and scopes is as follows.

Name	Description
phoenix	On-chain, crankless orderbook built on top of sokoban. We reviewed github.com/Ellipsis-Labs/phoenix-v1 commit `3d09bc8`.
phoenix-seat-manager	On-chain program, which automatically manages seats for markets on the Phoenix protocol. We reviewed github.com/Ellipsis-Labs/phoenix-seat-manager commit `e9630fc`
sokoban	Memory-efficient data structures library. For this engagement, we focused our analysis on the red black tree and node allocator. We reviewed github.com/Ellipsis-Labs/sokoban commit `9a7c2d0`.
ellipsis-macros	Miscellaneous macros for Solana program code, primarily intended for use by phoenix. We reviewed github.com/Ellipsis-Labs/ellipsis-macros commit `142c920`.
phoenix-sdk	Core SDK for interacting with the Phoenix on-chain order book, built on ellipsis-client. We reviewed github.com/Ellipsis-Labs/phoenix-sdk commit `a92a875`.
ellipsis-client	Lightweight unified interface around RPC and BanksClient. We reviewed github.com/Ellipsis-Labs/ellipsis-client commit `1b5168d`.

As part of this audit, we also provided proofs of concept to demonstrate certain scenarios. In particular, see our Adversarial Eviction POC.

03 | Discussion

As part of this engagement, we evaluated the on-chain program and data structures for a variety of issues. Drawing on our work with previous order books such as Serum, we are able to make important parallels to our past engagements. While we are unable to document all of our discussions, we include the important ones here.

Adversial Eviction

Part of our analysis focused on the design of the order book. One interesting feature which we analyzed heavily was eviction behavior when the order book was filled. We also provided a proof of concept to fully demonstrate this behavior in Adversarial Eviction POC.

This behavior is mitigated by two main factors. First, makers on the book need to have their seats explicitly reserved, making them semi-trusted. Second, phoenix allows for large configurations of up to 4096 orders, making clearing the order book relatively expensive for adversaries.

Data Structure Concerns

One subcomponent of this audit was ensuring that the data structures operated as intended. This has implications both for phoenix and also as an independent library. We sought to ensure that both use cases were sound.

Here we noted a critical issue in the Rust soundness of the red-black tree (OS-EPS-ADV-00). We also made suggestions around improving data structure efficiency (OS-EPS-SUG-01).

Denial of Service

We preface this section by noting that it is difficult to fully evaluate a program for denial of service issues. We applied a best-effort analysis to try and find critical areas where the program might not have performed sufficient validation of data inputs and unintentionally aborted. We noted two potential issues here, OS-EPS-ADV-02 and OS-EPS-ADV-03.

Solana Specific Issues

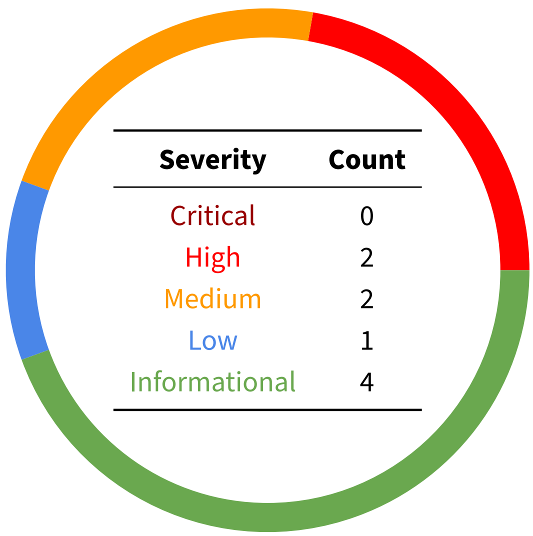
One other area we looked into was quirks with the Solana VM that the Ellipsis team overlooked or was unaware of. This includes various behaviors around account creation or reallocation, of which we've reported novel bugs around.

In particular, a quirk with Solana account creation ended up being the root cause of OS-EPS-ADV-02.

04 | Findings

Overall, we report 9 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 don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	2
Medium	2
Low	1
Informational	4

05 | 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 B.

ID	Severity	Status	Description
OS-EPS-ADV-00	High	Resolved	In Sokoban, the critbit, AVL tree, and red-black tree do not correctly implement Rust's DoubleEndedIterator trait.
OS-EPS-ADV-01	High	Resolved	Phoenix SDK parses all transactions, even those with errors. This can allow an attacker to spoof log transactions by manually calling the Phoenix program.
OS-EPS-ADV-02	Medium	Resolved	Edge case during account creation with extra lamports causes denial of service.
OS-EPS-ADV-03	Medium	Resolved	Overflows can occur during normal operation of the orderbook under certain parameter configurations.
OS-EPS-ADV-04	Low	Resolved	In the `phoenix-seat-manager`, lack of checks when claiming the market authority could potentially result in a seat eviction DOS vulnerability.

OS-EPS-ADV-00 [high] [resolved] | Invalid DoubleEndedIterator Trait Implementations

Description

The critbit, AVL tree, and red-black tree do not correctly implement Rust's `DoubleEndedIterator` trait, which is described here.

For instance, the red-black tree iterator's `next` and `next_back` method will cross each other, "double-counting" each element. This does not follow the spec, and can even be unsafe: when using `iter_mut`, one can obtain multiple mutable references to the same value.

```rust
let mut rbtree = RedBlackTree::<u64, u64, 100>::new();
rbtree.insert(0, 0);
rbtree.insert(1, 0);

let mut iter = rbtree.iter_mut();
let x: &mut u64 = iter.next().unwrap().1;
let y: &mut u64 = iter.next().unwrap().1;

*x = 1337;
assert_eq!(*y, 1337);
```

Remediation

Rewrite `next` and `next_back` so that they do not cross each other, or remove the implementations of `DoubleEndedIterator` altogether.

Patch

Resolved in #11.

OS-EPS-ADV-01 [high] [resolved] | SDK Transaction Spoofing

Description

When parsing events from transactions, the Phoenix SDK iterates over all the inner instructions to try and parse out `PhoenixInstruction::Log` instructions. Unfortunately, this loop fails to return when the transaction has errored, as specified in the `is_err` field.

sdk_client.rs RUST

```rust
for inner_ixs in tx.inner_instructions.iter() {
    for inner_ix in inner_ixs.iter() {
        let current_program_id = inner_ix.instruction.program_id.clone();
        if current_program_id != phoenix::id().to_string() {
            continue;
        }
        if inner_ix.instruction.data.is_empty() {
            continue;
        }
```

If a malicious user were to manually invoke the Phoenix program from a separate onchain program, inaccurate log events could be subsequently processed in `parse_phoenix_events`.

In conjunction with OS-EPS-SUG-02, this could lead to a denial of service condition for users of the SDK.

Remediation

Check if the transaction was successfully completed, and if not, skip processing of the transaction.

Patch

Resolved in #50.

OS-EPS-ADV-02 [med] [resolved] | Account Creation DOS

Description

Account creation primitives in phoenix will error if the account already has lamports.

This could, for example, allow an attacker to deny seat creation.

processor/manage$_seat.rs$ RUST

```rust
let space = size_of::<Seat>();
invoke_signed(
    &system_instruction::create_account(
        payer.key,
        seat.key,
        Rent::get()?.minimum_balance(space),
        space.try_into().unwrap(),
        &crate::ID,
    ),
    &[payer.clone(), seat.clone(), system_program.clone()],
    &[&[b"seat", market_key.as_ref(), trader.as_ref(), &[bump]]],
```

Remediation

Use transfer and allocate instead of `create_account` similar to what Anchor does.

RUST

```rust
// Fund the account for rent exemption.
// ...
// Allocate space.
// ...
// Assign to the spl token program.
```

Patch

Resolved in #1.

OS-EPS-ADV-03 [med] [resolved] | Explicate Overflow Boundaries

Description

Throughout phoenix, the largest numerical calculation occurs in the matching engine when calculating adjusted quote lots.

```rust
    inflight_order.adjusted_quote_lot_budget =
        inflight_order.adjusted_quote_lot_budget.saturating_sub(
            self.tick_size_in_quote_lots_per_base_unit
                * order_id.price_in_ticks
                * num_base_lots_quoted,
        );
```

Note that adjusted quote lots are declared as a `basic_u64_struct` with an internal maximum u64 representation.

```rust
macro_rules! basic_u64_struct {
    ($type_name:ident) => {
        #[derive(Debug, Clone, Copy, PartialOrd, Ord, Zeroable, Pod)]
        #[repr(transparent)]
        pub struct $type_name {
            inner: u64,
        }

        basic_u64!($type_name);
    };
}
```

More concretely, let

1. q be the number of quote atoms transacted
2. b be the number of decimals in the base token
3. $lots_q$ be quote atoms per lot
4. $lots_b$ be base atoms per lot

This calculation will abort if

$$q * lots_q * lots_b * 10^b \geq 2^{64}$$

Remediation

Because adjusted quote lots are multiplied by an additional factor of `base_lots_per_base_unit`, the maximum size can exceed the representable limit for u64.

Consider either increasing the internal representation for adjusted quote lots to u128 or explicating constraints on lot sizes.

Patch

The Ellipsis team acknowledges the issue and agrees to select parameters carefully with these constraints in mind. In particular, they note that upon large price fluctuations, new markets will likely be created, mitigating this issue for most practical usecases.

OS-EPS-ADV-04 [low] [resolved] | Seat Eviction DOS

Description

Upon claiming the market authority for a specified `Phoenix` market via the `phoenix-seat-manager` program, there is no check for the number of lamports in `seat_deposit_collector`.

This will cause subsequent executions of `EvictSeat` to fail due to missing funds.

Remediation

In `process_claim_market_authority` method, it should be verified that the `seat_deposit_collector` associated with the `market` possesses an adequate number of lamports to handle the rent for two `TokenAccount` for each seat currently occupied within the `market`.

Patch

Resolved in #15.

06 | 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 antipatterns and could lead to security issues in the future.

ID	Description
OS-EPS-SUG-00	Consider stronger enforcement of critical orderbook invariants.
OS-EPS-SUG-01	The red-black tree's node removal algorithm can be slightly improved.
OS-EPS-SUG-02	Consider using explicit error handling over hard panics
OS-EPS-SUG-03	Use checked truncation over potentially unsafe typecasts

OS-EPS-SUG-00 | Enforce Critical Orderbook Invariants

Description

Certain phoenix functions could use additional validation.

For example, `process_multiple_new_orders` could ensure that quote and base lots to deposit is equal to zero if there are no bids or asks respectively.

processor/new_order.rs RUST

```rust
if !bids.is_empty() {
    maybe_invoke_deposit(/**/)?;
} // else assert quote_lots_to_deposit == 0
if !asks.is_empty() {
    maybe_invoke_deposit(/**/)?;
} // else assert base_lots_to_deposit == 0
```

In `process_cancel_orders`, withdrawn quantities could similarly be asserted to zero if context is None.

processor/cancel_multiple_orders.rs RUST

```rust
if let Some(PhoenixVaultContext {
    // ...
}) = vault_context_option
{
    try_withdraw(/**/)?;
} // else assert num_base_lots_out == 0 && num_quote_lots_out == 0
```

Remediation

Consider adding relevant asserts to ensure critical orderbook invariants.

OS-EPS-SUG-01 | Red-Black Tree Optimization

As part of its balancing procedure, the red-black tree's `_remove_tree_node` method identifies a pivot node, which represents the subtree which has lost a black node. If the pivot node is also the root, the tree is already balanced. Otherwise, we invoke the `_fix_remove` method to balance the tree through rotations.

src/red_black_tree.rs RUST

```rust
if self.is_root(pivot_node_index) {
    self._color_black(pivot_node_index);
} else {
    self._fix_remove(pivot_node_index, parent_and_dir);
}
```

However, notice that if the pivot node is red, we can color it black to immediately balance the red-black tree. This case is ignored in `_fix_remove` because it immediately begins traversing up the tree.

Remediation

The `_remove_tree_node` should immediately color the node black if the pivot node is red.

RUST

```rust
if self.is_root(pivot_node_index) || self.is_red(pivot_node_index) {
    self._color_black(pivot_node_index);
} else {
    self._fix_remove(pivot_node_index, parent_and_dir);
}
```

OS-EPS-SUG-02 | Improved SDK Error Handling

In multiple areas in the Phoenix SDK, hard panics are used for error handling when encountering unexpected conditions.

sdk_client_core.rs RUST

```rust
        let header = match header_event {
            MarketEvent::Header { header } => Some(header),
            _ => {
                panic!("Expected a header event");
            }
        }?;
```

sdk_client_core.rs RUST

```rust
            }),
            _ => {
                panic!("Unexpected Event!");
            }
        }
```

As demonstrated in OS-EPS-ADV-04, some of these invariants may be violated.

Remediation

Manually log errors and return None instead of panicking.

OS-EPS-SUG-03 | Potentially Unsafe Truncation

Description

Phoenix uses unsafe typecasting to truncate integers. While we were unable to find a way to exploit these as is, it could lead to potentially unsafe behavior in a future refactor if integer bounds change.

markets/fifo.rs RUST

```rust
fn size_post_fee_adjustment(&self, size_in_adjusted_quote_lots:
    ↪ AdjustedQuoteLots) -> u64 {
    let fee_adjustment =
    ↪ self.compute_fee(AdjustedQuoteLots::MAX).as_u128() + u64::MAX as
    ↪ u128;
    (size_in_adjusted_quote_lots.as_u128() * u64::MAX as u128 /
    ↪ fee_adjustment) as u64
}
```

markets/fifo.rs RUST

```rust
fn compute_fee(&self, size_in_adjusted_quote_lots: AdjustedQuoteLots) ->
    ↪ AdjustedQuoteLots {
    AdjustedQuoteLots::new(
        ((size_in_adjusted_quote_lots.as_u128() * self.taker_fee_bps as
    ↪ u128 + 10000 - 1)
            / 10000) as u64,
    )
}
```

Remediation

Use safe casting variants such as `try_from` over potentially unsafe `as` casting.

A | Proofs of Concept

Adversial Eviction

RUST

```rust
fn test_malicious_eviction() {
    let mut empty_func = |_| {};
    let mut market = setup_market();
    let maker = 0;
    let trader = 1;

    for i in 1..5 {
        place_limit_order(
            &mut market,
            maker,
            100 + 10 * i,
            1000 + 100 * i,
            Side::Ask,
            &mut empty_func,
        )
        .unwrap();
    }
    print_ladder(&market);

    // place aggressive orders
    let mut order_ids = vec![];
    for _i in 0..BOOK_SIZE {
        if let (Some(order_id), _) =
            place_post_only_order(&mut market, trader, 105, 1, Side::Ask,
    ↪   &mut empty_func).unwrap()
        {
            order_ids.push(order_id);
        } else {
            panic!("unreachable");
        }
    }
    for order_id in order_ids {
        cancel_order(
            &mut market,
            trader,
            &order_id,
            Side::Ask,
```

```
                true,
                &mut empty_func,
            )
            .unwrap();
    }
    print_ladder(&market);
}
```


B | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities that immediately lead to 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 could lead to 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 could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation

C | 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 implementation of the program 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 sum, 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 get a comprehensive understanding of 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 other 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.