Econia Labs

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen r@osec.io

Naveen Kumar J naina@osec.io

Contents

01 | Executive Summary

Overview

Econia Labs engaged OtterSec to perform an assessment of the `econia` program. This assessment was conducted between November 21st and December 16th, 2022.

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 December 17th, 2022.

Key Findings

Over the course of this audit engagement, we produced 7 findings total.

In particular, we found that a critical verification check was missed while placing an order against the order book (OS-ECL-ADV-00). We also noted a low severity issue related to eviction design in the AVL queue, where an attacker can theoretically clear the orderbook (OS-ECL-ADV-01).

Additionally, we made recommendations around improved market order handling (OS-ECL-SUG-01, OS-ECL-SUG-00), recommendations for additional orderbook features (OS-ECL-SUG-02), along with recommendations for formal verification of the contracts (OS-ECL-VER-00, OS-ECL-VER-01).

Overall, we commend the Econia Labs team for being responsive and knowledgeable throughout the audit.

02 | Scope

The source code was delivered to us in a git repository at github.com/econia-labs/econia/tree/v4.0.0. This audit was performed against commit a62322a.

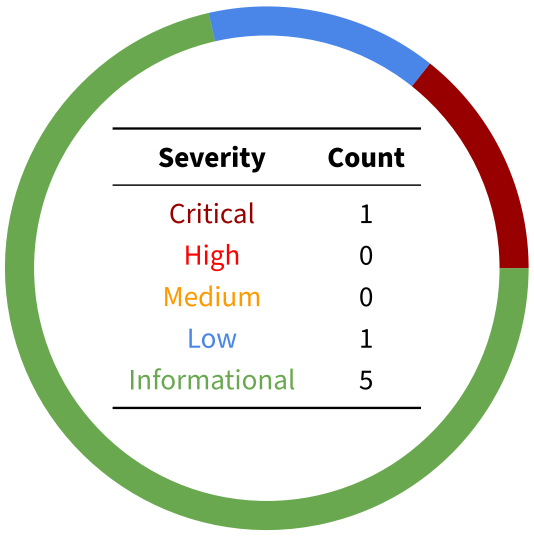
A brief description of the programs is as follows.

Name	Description
Econia	Hyper-parallelized on-chain orderbook for the Aptos blockchain.

03 | Findings

Overall, we report 7 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	1
High	0
Medium	0
Low	1
Informational	5

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 B.

ID	Severity	Status	Description
OS-ECL-ADV-00	Critical	Resolved	Missing types check against the market while placing an order leads to loss of funds.
OS-ECL-ADV-01	Low	Resolved	Forced eviction of legit orders when placed in bulk by an attacker.

OS-ECL-ADV-00 [crit] [resolved] | Missing Type Check While Placing Order

Description

In the functions `market::place_market_order` and `market::place_limit_order()`, when placing an order, there is no type verification against the original market types. Usually, the market should only allow orders of the same type, but this check was not enforced while placing an order. This would allow attackers to use incorrect coin types against the market, transferring coins of an incorrect type.

Proof of Concept

See Missing Type Check Bug POC.

Remediation

To mitigate this issue, add checks to verify if the passed types are the same as market types. This way, the operation aborts if someone tries to place orders with different types.

Patch

Patch added for both the functions in commit fcbfbcf8.

src/move/econia/sources/market.move DIFF

```diff
@@ -2327,6 +2345,10 @@ module econia::market {
         }

+    assert!(type_info::type_of<BaseType>() // Assert base type.
+         == order_book_ref_mut.base_type, E_INVALID_BASE);
+    assert!(type_info::type_of<QuoteType>() // Assert quote type.
+         == order_book_ref_mut.quote_type, E_INVALID_QUOTE);
```

OS-ECL-ADV-01 [low] | Forced Eviction Of Legit Orders

Description

AVL queue evicts orders when the tree exceeds a `CRITICAL_HEIGHT` or, when the number of active nodes becomes equal to `N_NODES_MAX`, to prevent excessive gas costs for insertion and deletion.

In theory, due to the limited orderbook capacity, an attacker can place enough orders to evict legitimate orders and then cancel these placed orders.

Proof of Concept

We profiled the gas required to fill up the whole tree and then replace all the legit orders from the tree. See Gas Profiling POC.

Before Patch:

- It took ~3 transactions to fill up the tree.
- To evict all existent legit orders it took ~6 transactions.

Remediation

One easy mitigation would be to increase the height of the tree. Currently, the `CRITICAL_HEIGHT` of the tree was passed as ‘10‘, which can hold ‘2048‘ unique priced orders.

Increasing the `CRITICAL_HEIGHT` of the tree can make it more expensive to evict all orders.

```rust
/// | Height        | Minimum size | Maximum size     |
/// |---------------|--------------|------------------|
/// | 10            | 232          | 2047             | <---
/// | 13            | 986          | 16383 (`n_max`)  |
/// | 18 (`h_max`)  | 10945        | 524287           |
```

Unfortunately, there is a hard limit on how much we can increase the tree height.

We profiled the eviction gas after the patch and it increases the number of transactions required by around 6x. While this isn’t perfect, it does represent a decent mitigation.

- It took ~20 transactions to fill up the tree.
- To evict all existent legit orders it took ~37 transactions.

As an additional mitigation, a minimum order and tick size should be chosen.

Patch

Patch Added in commit 9b3cada.

src/move/econia/sources/market.move DIFF

```diff
@@ -634,7 +634,7 @@ module econia::market {

-    const CRITICAL_HEIGHT: u8 = 10;
+     const CRITICAL_HEIGHT: u8 = 18;

}
```

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 antipatterns and could lead to security issues in the future.

ID	Description
OS-ECL-SUG-00	Add coin amount check to user::deposit_asset.
OS-ECL-SUG-01	Improve market order access control.
OS-ECL-SUG-02	Proposed improvements and new features for trading behavior

OS-ECL-SUG-00 | Add Coin Amount Check To User::Deposit_Asset

Description

In the function `user::deposit_asset`, if the asset type is a coin, check if the passed `optional_coins` is equal to the amount passed. This eliminates the assumption that the amount argument matches `coin.value` when the `optional_coins` argument is `option::some`.

Remediation

It is proposed that the function the asserts the coin value matches the passed amount when coins are `option::some` i.e when the asset type is not generic.

Patch

Patch in commit c5fa4d6d.

src/move/econia/sources/user.move DIFF

```diff
@@ -1801,14 +1798,18 @@ module econia::user {

        } else { // If asset is coin:
+            // Extract coins from option.
+            let coins = option::destroy_some(optional_coins);
+           // Assert passed amount matches coin value.
+           assert!(amount == coin::value(&coins),
     ↪   E_COIN_AMOUNT_MISMATCH);
```

OS-ECL-SUG-01 | Market Order Suggestions

1. Verify order access key when placing a limit order.

Description

To prevent a mismatch between a `market::Order.order_access_key` and the order access key used to look up a `user::Order`, it is proposed that:

(a) `user::place_order_internal` return the order access key from user-side insertion.

(b) `market::place_limit_order` verify the return of `user::place_order_internal` is equal to the local variable from `user::get_next_order_access_key_internal`.

Patch

Patch added in commit 171ae71

2. Check post-match size against min size rather than zero.

Description

Presently, after `market::place_limit_order` concludes optional cross-spread matching, the function returns if the remaining size to fill is zero.
Instead, it is proposed that the condition be modified to check that size is greater than the minimum order size for the market.

Patch

This was patched by checking the size against min size instead of zero. Corresponding commit 562bfea.

src/move/econia/sources/market.move DIFF

```diff
@@ -2268,8 +2270,11 @@ module econia::market  {

-         if ((restriction == IMMEDIATE_OR_CANCEL) || (size == 0))

+         if ((restriction == IMMEDIATE_OR_CANCEL) ||
+             (size < order_book_ref_mut.min_size))

}
```

3. Add `price` field to `market::Order`.

Description

For data locality and ease of indexing, it is proposed that a price field be added to `market::Order`.

The function `market::match` should be modified to incorporate a check that the price from the head of the AVL queue matches that inside the borrowed order as well as the function `market::place_limit_order` should be modified to specify the order price during AVL queue insertion.

Patch

Improvement was added in commit aeb53de.

OS-ECL-SUG-02 | Trading Improvements And Features

1. Support configurable self-match behaviour.

Description

Self-trading is not supported in the current version, the function `market::match` aborts if self-trading happens.

This might make it difficult for traders to accurately respond to fast-moving market conditions. It might be better to offer traders additional flexibility when it comes to self-trading options.

src/move/econia/sources/market.move RUST

```rust
fun match<
    BaseType,
    QuoteType
>([..]): ([..])
{
    [..]
    let (maker, custodian_id, size) =
        (order_ref_mut.user, order_ref_mut.custodian_id, fill_size);
    // Assert no self match.
    assert!(maker != taker, E_SELF_MATCH);
    [..]
}
```

Instead of aborting during self-trading, it is proposed that configurable self-match behaviour be supported. For example, allow the user to specify one of: abort, proceed, cancel, etc.

Patch

Feature added and discussion documented in #43.

2. Order state synchronization between the market side and the user side.

Description

The same order is stored in the `AVL queue` on the market side and `tablist` on the user side. It is suggested to verify critical fields like order size are equal between the two, this way we can ensure that the order is synchronized in both places without any discrepancy.

It is proposed to ensure synchronized order size between user-level and market-level states.

Patch

Improvement was added in commit 1eb831b.

3. Add support for passive restriction.

Description

Econia does not disclose the best bid/ask prices during normal operations, except when conducting mutation operations (such as taker or maker orders). This can make it hard for makers who want to place orders at some fixed or relative offset from the spread. It could make sense to add options that enable finer-grained relative pricing from the best bid/ask.

Patch

Feature added and discussion documented in #57.

06 | Formal Verification

Here, we present a discussion about the formal verification of smart contracts. We include example specifications, recommendations, and general ideas to formalize critical invariants.

We also note that including prover specifications in the Econia protocol would, at present, require stubbing out bitwise operations, which inform a substantial amount of code.

Seeing as the prover is not yet at a state where Econia production code could be run on it successfully, and seeing as the relevant specifications are covered by unit testing, it could make sense to wait until tooling is sufficient to support the full production codebase.

ID	Description
OS-ECL-VER-00	Specifications for the registry contract.
OS-ECL-VER-01	General specifications for the contracts.

OS-ECL-VER-00 | Registry Specifications

1. The value 0 corresponding to `CustodianCapability` and `UnderwriterCapability` are reserved and both follow 1-indexed ids. Consider using a specification to enforce this intended behavior.

src/move/econia/sources/registry.move RUST

```
spec CustodianCapability {
    invariant custodian_id != 0;
}

spec get_custodian_id {
    ensures result != 0;
}

spec UnderwriterCapability {
    invariant underwriter_id != 0;
}

spec get_underwriter_id {
    ensures result != 0;
}
```

2. To ensure that the registered market info is as expected, consider using a specification to enforce that the market parameters are valid. For example, the base and quote types must not be the same, and sizes must be valid.

src/move/econia/sources/registry.move RUST

```
spec MarketInfo {
    invariant base_type != quote_type;
    invariant lot_size   > 0         ;
    invariant tick_size  > 0         ;
    invariant min_size   > 0         ;
}
```

OS-ECL-VER-01 | General Specifications

1. Consider a specification that ensures the function `withdraw_utility_coins_all` properly withdraws all of the `UtilityCoins` from UtilityCoinStore, then deposits to `@econia`.

src/move/econia/sources/registry.move RUST

```rust
spec withdraw_utility_coins_all {
    ensures global<UtilityCoinStore<UtilityCoinType>>(
        resource_account::get_address()
    ).coins == coin::zero<UtilityCoinType>();
}
```

2. Consider a specification to check if the passed types are valid.

src/move/econia/sources/registry.move RUST

```rust
    spec <function> {
        aborts_if base_type != quote_type;
    }
```

A | Proofs of Concept

Missing Type Check Bug POC

RUST

```rust
##[test]
fun test_orders()

    [..]
    let side             = ASK; // Taker sell.
    let size_match       = MIN_SIZE_COIN + 36; // 40
    let size_post        = MIN_SIZE_COIN + 56; // 60
    let size             = size_match + size_post;  // 100
    let base_match       = size_match * LOT_SIZE_COIN;  // 40 * 2 = 80
    let base_post        = size_post * LOT_SIZE_COIN;   // 60 * 2 = 120
    let _base             = base_match + base_post;      // 200
    let price            = integrator_divisor * taker_divisor;  // 20
     ↪  * 5     = 100
    let quote_match      = size_match * price * TICK_SIZE_COIN; // 40 * 3
     ↪  * 100 = 12,000
    let quote_post       = size_post * price * TICK_SIZE_COIN;  // 60 * 3
     ↪  * 100 = 18,000
    let integrator_share = quote_match / integrator_divisor;    // 600
    let econia_share     = quote_match / taker_divisor -
     ↪  integrator_share;  // 1,800
    let fee              = integrator_share + econia_share; // 2400
    let quote_trade      = quote_match - fee;   // 9,600
    let _quote_total      = quote_trade + quote_post;    // 27,600
    let restriction      = NO_RESTRICTION;

    // Deposit coins
    user::deposit_coins<BC>(@user_0, MARKET_ID_COIN, NO_CUSTODIAN,
                            assets::mint_test(13377));
    user::deposit_coins<QC>(@user_0, MARKET_ID_COIN, NO_CUSTODIAN,
                            assets::mint_test(13377));

    user::deposit_coins<BC>(@user_1, MARKET_ID_COIN, NO_CUSTODIAN,
                            assets::mint_test(13377));
    user::deposit_coins<QC>(@user_1, MARKET_ID_COIN, NO_CUSTODIAN,
                            assets::mint_test(13377));
    let (market_order_id_0, _, _, _) = place_limit_order_user<BC, QC>(
        &user_0, MARKET_ID_COIN, @integrator, !side, size_match, price,
        restriction);

    assert!(is_list_node_order_active( // Assert order is active.
        MARKET_ID_COIN, !side, market_order_id_0), 0);

    // *** Placing Order With Incorrect Types ***
    place_limit_order_user<QC, QC>(
        &user_1, MARKET_ID_COIN, @integrator, side, size, price,
        restriction);

    assert!(user::get_collateral_value_simple_test<BC>(     // user0_base
    ↪   -> No change
        @user_0, MARKET_ID_COIN, NO_CUSTODIAN) == 13377, 0);

    assert!(user::get_collateral_value_simple_test<QC>(     //
    ↪   user0_quote -> deducted
        @user_0, MARKET_ID_COIN, NO_CUSTODIAN) < 13377, 0);

    assert!(user::get_collateral_value_simple_test<BC>(     // user1_base
    ↪   -> No change
        @user_1, MARKET_ID_COIN, NO_CUSTODIAN) == 13377, 0);

    assert!(user::get_collateral_value_simple_test<QC>(     //
    ↪   user1_quote  -> got added
        @user_1, MARKET_ID_COIN, NO_CUSTODIAN) > 13377, 0);

        // User -> Base (13377) -> Quote (13377)
// Expected
        // 0    -> +80          -> -12000   [BID]
        // 1    -> -80          -> + 9600   [ASK]
// Result
        // 0    -> 0            -> - 11920  [BID]
        // 1    -> 0            -> + 9,520  [ASK]
```

Eviction Gas POC

RUST

```rust
##[tokio::main]
async fn main() -> Result<(), Box<dyn Error>> {
    let mut harness = MoveHarness::new();
    let account_1 =
    ↪   &harness.new_account_at(AccountAddress::from_hex_literal("0x1337").unwrap());
    let move_addr = *account_1.address();
    println!(
        "{:?}",
        harness.publish_package(&account_1, Path::new("./move_old"))
    ↪   //Before Patch
    );

    // Fill nodes without eviction.
    // It takes 3 transactions to place orders without eviction. Gas for
    ↪   each transaction as follows
    // place_orders_poc used: 0 1823115
    // place_orders_poc used: 1 1894472
    // place_orders_poc used: 2 778645
    for i in 0..3_i64 {
        let test_name = "place_orders_poc";
        let order_count = 850;
        let max_orders_in_a_transaction = (order_count * i) + 1;
        let start = max_orders_in_a_transaction.to_le_bytes().to_vec();

        let end = if max_orders_in_a_transaction + order_count > 2049 {
            2049_u64.to_le_bytes().to_vec()
        } else { (max_orders_in_a_transaction +
    ↪   order_count).to_le_bytes().to_vec()};

        let data = vec![0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    ↪   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 55];
        let args = vec![&data[..], &start[..], &end[..]];

        let used_gas = eval_gas_with_real_params(
            &mut harness,
            &ModuleId::new(move_addr, Identifier::new("market")?),
            &test_name ,
            vec![],
            args,
        );
        println!("{test_name} used: {} {:?}", i, used_gas);
    }
    // Filling up evicting old orders.
    // It took 6 transactions to remove all previous orders and
    ↪  transaction gas costs as follows
    // place_orders_poc used: 3 1523348
    // place_orders_poc used: 4 1452269
    // place_orders_poc used: 5 1873712
    // place_orders_poc used: 6 1832716
    // place_orders_poc used: 7 1727444
    // place_orders_poc used: 8 295998
    for i in 3..9_i64 {
        let test_name = "place_orders_poc";
        let order_count = 550;
        let max_orders_in_a_transaction = (order_count * i) + 1;
        let start = max_orders_in_a_transaction.to_le_bytes().to_vec();

        let end = if max_orders_in_a_transaction + order_count > 4495 {
            4495_u64.to_le_bytes().to_vec()
        } else { (max_orders_in_a_transaction +
    ↪   order_count).to_le_bytes().to_vec()};

        let data = vec![0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    ↪   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 55];
        let args = vec![&data[..], &start[..], &end[..]];


        let used_gas = eval_gas_with_real_params(
            &mut harness,
            &ModuleId::new(move_addr, Identifier::new("market")?),
            &test_name ,
            vec![],
            args,
        );
        println!("{test_name} used: {} {:?}", i, used_gas);

    }
}
```

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