Security Assessment Report

Monaco Protocol v0.2.1

November 21st, 2022

Summary

The sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the Monaco Protocol v0.2.1 Solana smart contract programs. The artifact of the audit was the source code of the following on-chain smart contracts excluding tests in a private repository.

- **Contract "monaco_protocol":**
    - v0.2.1 - Commit `987a4c29a5f56f473dfc326d53f9a7a4d40be21b`

The audit revealed 17 issues or questions. After the initial review, the source code was moved to a new repository at https://github.com/MonacoProtocol/protocol. The team responded with the following commits for the post-audit review, which is to validate if the reported issues have been addressed.

- v0.5.0 - Commit `ac86fc54ff87195de6a69397fd15beb2c01ff8fe`

This report describes the findings and resolutions in detail.

Table of Contents

Methodology and Scope of Work

The sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in Solana smart contract security, program analysis, testing and formal verification, performed a comprehensive manual code review, software static analysis and penetration testing.

Assisted by the sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
    - Missing ownership checks
    - Missing signer checks
    - Signed invocation of unverified programs
    - Solana account confusions
    - Arithmetic over- or underflows
    - Numerical precision errors
    - Loss of precision in calculation
    - Insufficient SPL-Token account verification
    - Missing rent exemption assertion
    - Casting truncation
    - Did not follow security best practices
    - Outdated dependencies
    - Redundant code
    - Unsafe Rust code
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

Result Overview

In total, the audit team found the following issues.

MONACO PROTOCOL v0.2.1

Issue	Impact	Status
[M-1] Insufficiently validated market_outcome	Medium	Fixed
[M-2] Unvalidated market_matching_pool in CancelBet	Medium	Fixed
[M-3] Insufficiently validated bet orders, the outcome and matching pools	Medium	Fixed
[M-4] Double bet order removal	Medium	Fixed
[M-5] Integer overflows	Medium	Fixed
[L-1] Inconsistent behaviors when creating and closing matching pools	Low	Fixed
[L-2] Inconsistent comments and implementations	Low	Fixed
[L-3] The outcome index provided by users is not checked	Low	Fixed
[L-4] Operator type string validation	Low	Fixed
[L-5] Market parameters and unused fields	Low	Fixed
[L-6] Odds truncation	Low	Fixed
[I-1] Removing all admin operators is allowed	Informational	Fixed
[I-2] Unrestricted market and account closures	Informational	Fixed
[I-3] The “matches” field in “BetOrder” is not used	Informational	Fixed
[I-4] Market title is included in market PDA seeds and can be modified	Informational	Fixed
[I-5] Redundant market status check	Informational	Fixed
[I-6] Design questions	Informational	Fixed

Findings in Detail

IMPACT – MEDIUM

[M-1] Insufficiently validated market_outcome

A market may have multiple `market_outcome` and the `data.market_outcome_index` should be associated with `market_outcome`, given they are both provided by the user.

However, the validation only checks if the `market_outcome` is associated with the market but doesn't enforce it's bounded by the outcome index via the outcome title.

Since the accounting is done solely based on the market outcome index, this only affects the internal market outcome and matching pool stats.

```
/* programs/betdex_core/src/context.rs */
011 | #[derive(Accounts)]
012 | #[instruction(_distinct_seed: String, data: BetOrderData)]
013 | pub struct CreateBet<'info> {
060 |     #[account(mut, has_one = market)]
061 |     pub market_outcome: Account<'info, MarketOutcome>,
075 | }
```

Resolution

Now `market_outcome` and the market outcome index are bounded via PDA validation. This issue has been resolved.

IMPACT – MEDIUM

[M-2] Unvalidated market_matching_pool in CancelBet

The market_matching_pool is not validated. Since removing a bet order that is not in the matching_pool.bet_orders is allowed (as shown in the test case below), it's possible to provide a matching pool that is not related to the current market/order and manipulate its matching_pool.liquidity_amount.

```
/* programs/betdex_core/src/context.rs */
077 | #[derive(Accounts)]
078 | pub struct CancelBet<'info> {
079 |     #[account(mut)]
080 |     pub bet_order: Account<'info, BetOrder>,
091 |     #[account(mut, address = bet_order.market @ CoreError::CancelationMarketMismatch)]
092 |     pub market: Account<'info, Market>,
093 |     #[account(mut)]
094 |     pub market_matching_pool: Account<'info, MarketMatchingPool>,
108 | }

/* programs/betdex_core/src/lib.rs */
083 | pub fn cancel_bet(ctx: Context<CancelBet>) -> Result<()> {
088 |     instructions::matching::matching_pool::update_on_cancel(
089 |         bet_order,
090 |         &mut ctx.accounts.market_matching_pool,
091 |     )?;
118 | }

/* programs/betdex_core/src/instructions/matching/matching_pool.rs */
104 | pub fn update_on_cancel(
105 |     bet_order: &Account<BetOrder>,
106 |     matching_pool: &mut MarketMatchingPool,
107 | ) -> Result<()> {
109 |     matching_pool.liquidity_amount = matching_pool
110 |         .liquidity_amount
111 |         .checked_sub(bet_order.voided_stake)
112 |         .ok_or(CoreError::MatchingLiquidityAmountUpdateError)?;
113 |     matching_pool.bet_orders.remove_item(&bet_order.key());
114 |
115 |     Ok(())
116 | }
```

Test case: removing a key that is not in the queue.

```
#[test]
fn test_remove_not_exist() {
    let queue = &mut generate_populated_queue(4, 3);
    let to_remove = Pubkey::new_unique();
    println!("[before] queue.len() = {}", queue.len());
    print_queue(&queue.items);
    queue.remove_item(&to_remove);
    println!("[after rm {}] queue.len() = {}", to_remove, queue.len());
    print_queue(&queue.items);
    assert_eq!(3, queue.len());
}
```

The output:

```
running 1 test
[before] queue.len() = 3
-----------------------------------------
[0] 4uQeVj5tqViQh7yWWGStvkEG1Zmhx6uasJtWCJziofM
[1] 8opHzTAnfzRpPEx21XtnrVTX28YQuCpAjcn1PczScKh
[2] CiDwVBFgWV9E5MvXWoLgnEgn2hK7rJikbvfWavzAQz3
[3] 11111111111111111111111111111111
-----------------------------------------

[after rm GcdayuLaLyrdmUu324nahyv33G5poQdLUEZ1nEytDeP] queue.len() = 3
-----------------------------------------
[0] 4uQeVj5tqViQh7yWWGStvkEG1Zmhx6uasJtWCJziofM
[1] 8opHzTAnfzRpPEx21XtnrVTX28YQuCpAjcn1PczScKh
[2] CiDwVBFgWV9E5MvXWoLgnEgn2hK7rJikbvfWavzAQz3
[3] 11111111111111111111111111111111
-----------------------------------------
test state::market_account::tests::test_remove_not_exist ... ok
```

Resolution

The PDA validation of `market_matching_pool` has been added. Since its seed contains the market, the outcome index, price and the backing direction, this makes sure the provided `market_matching_pool` is consistent.

IMPACT – MEDIUM

[M-3] Insufficiently validated bet orders, the outcome and matching pools

The bet_order_back.market_outcome_index is not validated against the market_outcome. As a result, it's possible to provide a different market_outcome associated with the given market such that the matching pools are inconsistent with the bet orders.

```
/* programs/betdex_core/src/context.rs */
173 | #[derive(Accounts)]
174 | pub struct MatchBets<'info> {
181 |     pub bet_order_lay: Account<'info, BetOrder>,
188 |     #[account(
190 |         seeds = [
192 |             market_outcome.title.as_ref(),
198 |     pub market_matching_pool_lay: Account<'info, MarketMatchingPool>,

206 |     pub bet_order_back: Account<'info, BetOrder>,
213 |     #[account(
215 |         seeds = [
217 |             market_outcome.title.as_ref(),
223 |     pub market_matching_pool_back: Account<'info, MarketMatchingPool>,

226 |     #[account(mut, has_one = market @ CoreError::MatchingMarketMismatch)]
227 |     pub market_outcome: Account<'info, MarketOutcome>,
250 | }

/* programs/betdex_core/src/instructions/matching/matching_one_to_one.rs */
010 | pub fn match_bets(ctx: &mut Context<MatchBets>) -> Result<()> {
022 |     require!(
023 |         back.market_outcome_index == lay.market_outcome_index,
024 |         CoreError::MatchingMarketOutcomeMismatch
025 |     );
```

Resolution

The PDA validation of market_outcome has been added. Since its seed contains the market the matching market outcome index, this makes sure the provided market_outocme is consistent. This issue has been fixed.

IMPACT – MEDIUM

[M-4] Double bet order removal

When removing the first or the last item in the queue twice, an unmatched item is incorrectly removed in the second removal attempt,

1. Remove the first item twice

```
fn print_queue(queue:&Vec<Pubkey>) {
    let len = queue.len();
    let mut i = 0;
    println!("-----------------------------------------");
    while i < len {
        println!("[{}] {}", i, queue[i]);
        i += 1;
    }
    println!("-----------------------------------------");
}

#[test]
fn test_double_remove_1() {
    // remove front index 0
    let queue = &mut generate_populated_queue(4, 3);
    let to_remove = queue.items[0];
    println!("[before] queue.len() = {}", queue.len());
    print_queue(&queue.items);
    queue.remove_item(&to_remove);
    println!("[after rm {}] queue.len() = {}", to_remove, queue.len());
    print_queue(&queue.items);
    queue.remove_item(&to_remove);
    println!("[after rm {}] queue.len() = {}", to_remove, queue.len());
    print_queue(&queue.items);
    assert_eq!(2, queue.len());
}
```

Output

```
running 1 test
[before] queue.len() = 3
-----------------------------------------
[0] 4uQeVj5tqViQh7yWWGStvkEG1Zmhx6uasJtWCJziofM
[1] 8opHzTAnfzRpPEx21XtnrVTX28YQuCpAjcn1PczScKh
```

```
[2] CiDwVBFgWV9E5MvXWoLgnEgn2hK7rJikbvfWavzAQz3
[3] 11111111111111111111111111111111
----------------------------------------

[enter remove_item] len = 3, front_index = 0, last_index = 2
 - Found-1. front_index = 0, relative_index = 0, index = 0
[after rm 4uQeVj5tqViQh7yWWGStvkEG1Zmhx6uasJtWCJziofM] queue.len() = 2
----------------------------------------
[0] 4uQeVj5tqViQh7yWWGStvkEG1Zmhx6uasJtWCJziofM
[1] 8opHzTAnfzRpPEx21XtnrVTX28YQuCpAjcn1PczScKh
[2] CiDwVBFgWV9E5MvXWoLgnEgn2hK7rJikbvfWavzAQz3
[3] 11111111111111111111111111111111
----------------------------------------

[enter remove_item] len = 2, front_index = 1, last_index = 2
 - idx_0_to_last.len = 3
 - Found-2. index = 0
[after rm 4uQeVj5tqViQh7yWWGStvkEG1Zmhx6uasJtWCJziofM] queue.len() = 1
----------------------------------------
[0] 8opHzTAnfzRpPEx21XtnrVTX28YQuCpAjcn1PczScKh
[1] CiDwVBFgWV9E5MvXWoLgnEgn2hK7rJikbvfWavzAQz3
[2] CiDwVBFgWV9E5MvXWoLgnEgn2hK7rJikbvfWavzAQz3
[3] 11111111111111111111111111111111
----------------------------------------

thread 'state::market_account::tests::test_double_remove_1' panicked at 'assertion failed: `(left ==
right)`
  left: `2`,
 right: `1`', programs/betdex_core/src/state/market_account.rs:261:9
```

2. Remove the last item twice

```
#[test]
fn test_double_remove_2() {
    // remove the last bet order index 4
    let queue = &mut generate_populated_queue(8, 5);
    let to_remove = queue.items[4];
    println!("[before] queue.len() = {}", queue.len());
    queue.remove_item(&to_remove);
    println!("[after 1st-rm] queue.len() = {}", queue.len());
    queue.remove_item(&to_remove);
    println!("[after 2nd-rm] queue.len() = {}", queue.len());
    assert_eq!(4, queue.len());
}
```

Output

```
running 1 test
[before] queue.len() = 3
-----------------------------------------
[0] 4uQeVj5tqViQh7yWWGStvkEG1Zmhx6uasJtWCJziofM
[1] 8opHzTAnfzRpPEx21XtnrVTX28YQuCpAjcn1PczScKh
[2] CiDwVBFgWV9E5MvXWoLgnEgn2hK7rJikbvfWavzAQz3
[3] 11111111111111111111111111111111
-----------------------------------------

[enter remove_item] len = 3, front_index = 0, last_index = 2
 - Found-1. front_index = 0, relative_index = 2, index = 2
[after rm CiDwVBFgWV9E5MvXWoLgnEgn2hK7rJikbvfWavzAQz3] queue.len() = 2
-----------------------------------------
[0] 4uQeVj5tqViQh7yWWGStvkEG1Zmhx6uasJtWCJziofM
[1] 8opHzTAnfzRpPEx21XtnrVTX28YQuCpAjcn1PczScKh
[2] CiDwVBFgWV9E5MvXWoLgnEgn2hK7rJikbvfWavzAQz3
[3] 11111111111111111111111111111111
-----------------------------------------

[enter remove_item] len = 2, front_index = 0, last_index = 1
 - idx_0_to_last.len = 2
 - Found-3. front_index = 0, relative_index = 2, index = 2
[after rm CiDwVBFgWV9E5MvXWoLgnEgn2hK7rJikbvfWavzAQz3] queue.len() = 1
-----------------------------------------
[0] 8opHzTAnfzRpPEx21XtnrVTX28YQuCpAjcn1PczScKh
[1] 8opHzTAnfzRpPEx21XtnrVTX28YQuCpAjcn1PczScKh
[2] 11111111111111111111111111111111
[3] 4uQeVj5tqViQh7yWWGStvkEG1Zmhx6uasJtWCJziofM
-----------------------------------------

thread 'state::market_account::tests::test_double_remove_2' panicked at 'assertion failed: `(left ==
right)`
  left: `2`,
 right: `1`', programs/betdex_core/src/state/market_account.rs:278:9
```

Resolution

This issue has been fixed.

IMPACT – MEDIUM

[M-5] Integer overflows

```
/* programs/betdex_core/src/instructions/market/update_market.rs */
017 | fn update_market(market: &mut Market, title: String, lock_time: i64, now: i64) -> Result<()> {
018 |     if (lock_time - now) < 0 {  // lock_time is provided by users
023 |         return Err(error!(CoreError::LockTimeInvalid));
024 |     }
```

```
/* programs/betdex_core/src/state/market_account.rs */
138 | fn back(&self) -> u32 {
139 |     (self.front + self.len) % self.size()
140 | }
```

```
/* programs/betdex_core/src/instructions/matching/matching_pool.rs */
056 | pub fn update_matching_queue_with_new_bet(
057 |     market_matching_pool: &mut Account<MarketMatchingPool>,
058 |     bet_order_account: &Account<BetOrder>,
059 | ) -> Result<()> {
060 |     market_matching_pool.liquidity_amount += bet_order_account.stake;
067 | }
```

```
/* programs/betdex_core/src/instructions/bet_order/match_bet_order.rs */
031 | fn match_bet_order_internal(bet_order: &mut BetOrder, stake_matched: u64, odds_matched: f64) {
032 |     if stake_matched <= bet_order.stake_unmatched {
            // safe: caller makes sure bet_order.stake_unmatched >= stake_matched
034 |         bet_order.stake_unmatched -= stake_matched;
            // u64, may overflow
035 |         bet_order.payout += calculate_backing_payout(stake_matched, odds_matched);
036 |     }
037 | }
```

Resolution

The runtime overflow check is enabled. This issue has been fixed.

IMPACT – LOW

[L-1] Inconsistent behaviors when creating and closing matching pools

1. Create the matching pool(s)

Since `create_bet()` can just create one `market_matching_pool` of the pool pair when the pool doesn't exist. If a pool is firstly created by `create_bet()`, it may not be possible to create the pool pair via `initialize_market_matching_pool()` since one of them has already been created.

```
/* programs/betdex_core/src/context.rs */
011 | #[derive(Accounts)]
012 | #[instruction(_distinct_seed: String, data: BetOrderData)]
013 | pub struct CreateBet<'info> {
047 |     #[account(
048 |         init_if_needed,
049 |         seeds = [
050 |             market.key().as_ref(),
051 |             market_outcome.title.as_ref(),
052 |             format!("{:.3}", data.odds).as_ref(),
053 |             data.backing.to_string().as_ref()
054 |         ],
058 |     )]
059 |     pub market_matching_pool: Box<Account<'info, MarketMatchingPool>>,
075 | }

/* programs/betdex_core/src/context.rs */
328 | #[derive(Accounts)]
329 | #[instruction(odds: f64)]
330 | pub struct InitializeMarketMatchingPool<'info> {
337 |     #[account(
338 |         init,
339 |         seeds = [
340 |             market.key().as_ref(),
341 |             market_outcome.title.as_ref(),
342 |             format!("{:.3}", odds).as_ref(),
343 |             true.to_string().as_ref(),
344 |         ],
348 |     )]
349 |     pub market_matching_pool_back: Account<'info, MarketMatchingPool>,
350 |     #[account(
351 |         init,
352 |         seeds = [
353 |             market.key().as_ref(),
354 |             market_outcome.title.as_ref(),
355 |             format!("{:.3}", odds).as_ref(),
356 |             false.to_string().as_ref(),
357 |         ],
```

```
361 |     )]
362 |     pub market_matching_pool_lay: Account<'info, MarketMatchingPool>,
368 | }
```

2. Close the matching pool pair

For a given combination of `market`, `market_outcome.title` and `_odds`, are there always matching pool pairs (`market_matching_pool_back` and `market_matching_pool_lay`)? This is true if the pools are created by `initialize_market_matching_pool()`. However, it may not hold.

```
/* programs/betdex_core/src/context.rs */
370 | #[derive(Accounts)]
371 | #[instruction(_odds: f64)]
372 | pub struct CloseMarketMatchingPool<'info> {
380 |     #[account(
381 |         mut,
382 |         seeds = [
383 |             market.key().as_ref(),
384 |             market_outcome.title.as_ref(),
385 |             format!("{:.3}", _odds).as_ref(),
386 |             true.to_string().as_ref(),
387 |         ],
390 |         close = purchaser,
391 |     )]
392 |     pub market_matching_pool_back: Account<'info, MarketMatchingPool>,
393 |     #[account(
394 |         mut,
395 |         seeds = [
396 |             market.key().as_ref(),
397 |             market_outcome.title.as_ref(),
398 |             format!("{:.3}", _odds).as_ref(),
399 |             false.to_string().as_ref(),
400 |         ],
403 |         close = purchaser,
404 |     )]
405 |     pub market_matching_pool_lay: Account<'info, MarketMatchingPool>,
412 | }
```

Resolution

Instruction `initialize_market_matching_pool` has been removed and it's possible to close one matching pool. These issues have been fixed.

IMPACT – LOW

[L-2] Inconsistent comments and implementations

The comment says "use odds of a newer bet-order". However, the implementation is the opposite. The `selected_odds` is the odds of an older bet order.

```
/* programs/betdex_core/src/instructions/matching/matching_one_to_one.rs */
010 | pub fn match_bets(ctx: &mut Context<MatchBets>) -> Result<()> {
036 |     // use odds of a newer bet-order
037 |     let selected_odds = if back.creation_timestamp < lay.creation_timestamp {
038 |         back.expected_odds
039 |     } else {
040 |         lay.expected_odds
041 |     };
105 | };

/* programs/betdex_core/src/instructions/matching/matching_pool.rs */
007 | pub fn update_on_match(
015 | ) -> Result<()> {
025 |     if stake_matched > 0_u64 {
026 |         market_outcome.latest_matched_odds =
027 |             if back_bet_order.creation_timestamp < lay_bet_order.creation_timestamp {
028 |                 back_bet_order.expected_odds
029 |             } else {
030 |                 lay_bet_order.expected_odds
031 |             };
037 |     }
```

Resolution

This issue has been fixed.

IMPACT – LOW

[L-3] The outcome index provided by users is not checked

```
/* programs/betdex_core/src/instructions/bet_order/create_bet_order.rs */
009 | pub fn create_bet_order(
010 |     bet_order: &mut Account<BetOrder>,
014 |     data: &BetOrderData,
015 | ) -> Result<()> {
040 |     bet_order.market_outcome_index = data.market_outcome_index;
055 | }
```

```
/* programs/betdex_core/src/lib.rs */
329 | pub fn settle_market(
330 |     ctx: Context<UpdateMarketStatus>,
331 |     winning_outcome_index: u16,
332 | ) -> Result<()> {
337 |     instructions::market::settle(ctx, winning_outcome_index)
338 | }
```

The index should be checked against the market.market_outcomes.len()

Resolution

The index is checked by PDA. This issue has been fixed.

IMPACT – LOW

[L-4] Operator type string validation

When validating the operator_type, it first converts the string to the upper-case version. As a result, "CRANK" and "CRanK" can both pass the validation. However, the PDA seeds expect "CRANK" instead of "CRanK".

```
/* programs/betdex_core/src/context.rs */
126 | #[derive(Accounts)]
127 | #[instruction(operator_type: String)]
128 | pub struct AuthoriseOperator<'info> {
129 |     #[account(
130 |         init_if_needed,
131 |         seeds = [operator_type.as_ref()],
135 |     )]
143 | }

/* programs/betdex_core/src/instructions/operator.rs */
009 | pub fn authorise_operator(
013 |     operator_type: String,
014 | ) -> Result<()> {
015 |     validate_operator_type(operator_type)?;
022 | }

/* programs/betdex_core/src/instructions/operator.rs */
045 | fn validate_operator_type(operator_type: String) -> Result<()> {
046 |     let result = OperatorType::from_str(&*operator_type.to_uppercase());
047 |     require!(result.is_ok(), CoreError::InvalidOperatorType);
048 |     Ok(())
049 | }
```

Resolution

This issue has been fixed.

IMPACT – LOW

[L-5] Market parameters and unused fields

1. Validate parameters when creating markets. For example, what if there are duplications in `market_outcomes`

```
/* programs/betdex_core/src/lib.rs */
260 | pub fn create_market(
261 |     ctx: Context<CreateMarket>,
262 |     event_account: Pubkey,
263 |     title: String,
264 |     market_type: String,
265 |     market_outcomes: Vec<String>,
266 |     market_lock_timestamp: i64,
267 |     max_decimals: u8,
268 | ) -> Result<()> {
286 | }
```

2. Unused fields

```
/* programs/betdex_core/src/state/market_account.rs */
003 | #[account]
004 | pub struct Market {
005 |     pub authority: Pubkey,
006 |     pub event_account: Pubkey,
012 |     pub published: bool,
017 |     pub market_type: String,
021 |     pub title: String,
024 | }
```

`market.authority` is not used. The authorization is done by checking if the signer is an authorized operator. Similarly, `event_account`, `published`, `market_type` and `title` are assigned but not used.

3. In general, the outcome strings are not associated with `MarketOutcome` structures.

Resolution

These issues have been fixed.

IMPACT – LOW

[L-6] Odds truncation

When creating matching pools, the odds are truncated using format!("{:.3}", data.odds). However, when maintaining the odds whitelist in the market outcome, the odds are not truncated. As a result, it's possible that different bet orders with different odds will be grouped in a same matching pool. Is it an intended behavior?

```
/* programs/betdex_core/src/context.rs */
012 | #[instruction(_distinct_seed: String, data: BetOrderData)]
013 | pub struct CreateBet<'info> {
047 |     #[account(
048 |         init_if_needed,
049 |         seeds = [
052 |             format!("{:.3}", data.odds).as_ref(),
054 |         ],
058 |     )]
059 |     pub market_matching_pool: Box<Account<'info, MarketMatchingPool>>,

/* programs/betdex_core/src/instructions/bet_order/create_bet_order.rs */
009 | pub fn create_bet_order(
015 | ) -> Result<()> {
033 |     require!(
034 |         market_outcome.odds_ladder.contains(&data.odds),
036 |     );

#[test]
fn test_odds_format() {
    let odds_1 = 0.1234_f64;
    let odds_2 = 0.1235_f64;
    let odds_3 = 0.1235001_f64;
    let odds_4 = 0.1236_f64;
    println!("odds_1 = {}, format odds_1 = {}", odds_1, format!("{:.3}", odds_1));
    println!("odds_2 = {}, format odds_2 = {}", odds_2, format!("{:.3}", odds_2));
    println!("odds_3 = {}, format odds_3 = {}", odds_3, format!("{:.3}", odds_3));
    println!("odds_4 = {}, format odds_4 = {}", odds_4, format!("{:.3}", odds_4));
}
// ----------------------------------------
// odds_1 = 0.1234,    format odds_1 = 0.123
// odds_2 = 0.1235,    format odds_2 = 0.123
// odds_3 = 0.1235001, format odds_3 = 0.124
// odds_4 = 0.1236,    format odds_4 = 0.124
```

Resolution

The issue has been fixed. Now 3-decimal rule is enforced when adding prices.

IMPACT – INFO

[I-1] Removing all admin operators is allowed

It's possible to remove all admin operators through `remove_authorised_operator()`.

Once this happens, anyone can re-init the authorized operators including admins.

Resolution

The issue has been fixed.

IMPACT – INFO

[I-2] Unrestricted market and account closures

1. An market operator can close any market by calling via `close_market()` instruction without any market status checks, what if being called by accident?

2. `close_account` instruction can be used to close any account created by the contract without checking status. Although it can only be used by admin operators, it's risky if a wrong account is provided.

Resolution

The issue has been fixed.

IMPACT – INFO

[I-3] The “matches” field in “BetOrder” is not used

```
/* programs/betdex_core/src/state/bet_order_account.rs */
023 | #[account]
024 | pub struct BetOrder {
037 |     pub matches: Vec<BetOrderMatch>,
038 | }
```

Resolution

The issue has been fixed.

IMPACT – INFO

[I-4] Market title is included in market PDA seeds and can be modified

The market title is included in the market PDA seed and is allowed to be changed via the update market instruction. Is this an intended behavior?

```
/* programs/betdex_core/src/context.rs */
287 | #[derive(Accounts)]
288 | #[instruction(event_account: Pubkey, title: String)]
289 | pub struct CreateMarket<'info> {
290 |     #[account(
291 |         init,
292 |         seeds = [
293 |             event_account.as_ref(),
294 |             title.as_ref()
295 |         ],
299 |     )]
300 |     pub market: Account<'info, Market>,
326 | }

/* programs/betdex_core/src/lib.rs */
321 | pub fn update_market(ctx: Context<UpdateMarket>, title: String, lock_time: i64) -> Result<()> {
326 |     instructions::market::update(ctx, title, lock_time)
327 | }

/* programs/betdex_core/src/instructions/market/update_market.rs */
010 | pub fn update(ctx: Context<UpdateMarket>, title: String, lock_time: i64) -> Result<()> {
011 |     let market = &mut ctx.accounts.market;
014 |     update_market(market, title, lock_time, now)
015 | }
016 |
017 | fn update_market(market: &mut Market, title: String, lock_time: i64, now: i64) -> Result<()> {
026 |     if !title.is_empty() && !market.title.eq(&title) {
027 |         market.title = title;
028 |     }
035 | }
```

Resolution

The issue has been fixed.

IMPACT – INFO

[I-5] Redundant market status check

The `status` cannot be `Open` and `Locked` at the same time.

```
/* programs/betdex_core/src/instructions/bet_order/create_bet_order.rs */
057 | fn validate_market_for_bet_order(market: &Market, now: UnixTimestamp) -> Result<()> {
059 |     let status = &market.market_status;
061 |     require!(
062 |         status == &MarketStatus::Open,
063 |         CoreError::CreationMarketNotOpen
064 |     );
073 |     require!(
074 |         *market_lock_timestamp > now && *status != MarketStatus::Locked,
075 |         CoreError::CreationMarketLocked
076 |     );
079 | }
```

Resolution

The issue has been fixed.

IMPACT – INFO

[I-6] Design questions

1. The market close operator can be different from the one who created the market. Is this an intended behavior?

2. Why does the matched bet order have to be the first order in the match pool? Is this the intended behavior?

```
/* programs/betdex_core/src/instructions/matching/matching_pool.rs */
069 | fn update_matching_queue_with_matched_bet(
070 |     matching_pool: &mut MarketMatchingPool,
071 |     amount_matched: u64,
072 |     matched_bet: Pubkey,
073 |     fully_matched: bool,
074 | ) -> Result<()> {
084 |     if fully_matched {
085 |         remove_bet_order_from_matching_queue(&mut matching_pool.bet_orders, matched_bet)?;
086 |     }
087 |
088 |     Ok(())
089 | }
090 |
091 | pub fn remove_bet_order_from_matching_queue(
092 |     bet_order_queue: &mut Cirque,
093 |     bet_order: Pubkey,
094 | ) -> Result<()> {
095 |     let removed_item = bet_order_queue.dequeue();
096 |     require!(removed_item.is_some(), CoreError::MatchingQueueIsEmpty);
097 |     require!(
098 |         bet_order == removed_item.unwrap(),
099 |         CoreError::IncorrectBetOrderDequeueAttempt
100 |     );
101 |     Ok(())
102 | }
```

Resolution

The issue has been fixed.

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a sec3 (the "Company") and BetDEX Labs (the "Client"). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, sec3 (formerly Soteria) is a leading blockchain security company that currently focuses on Solana programs. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.

sec3™