sec3™

Security Assessment Report

Monaco Protocol v0.9.0

May 25, 2023

Summary

The sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the Monaco Protocol Solana smart contract at https://github.com/MonacoProtocol/protocol. The initial audit was done on the source code of the following version

- **Contract "monaco_protocol":**
  - v0.9.0, commit e9402f2f0fe08d42248e8eb3d949f89a5609f9cf

The review revealed 6 issues or questions, which have been resolved.

Table of Contents

Result Overview 3
Findings in Detail 4
[M-1] Unprocessed delay-expired orders 4
[L-1] Add Voided to the TRANSFER_SURPLUS_ALLOWED_STATUSES 5
[I-1] inplay not explicitly initialized 6
[I-2] Use market status to safeguard settle_market_position 7
[Q-1] Is inplay_order_delay:u8 (255 seconds) large enough? 8
[Q-2] move_market_to_inplay range check 9
Appendix: Methodology and Scope of Work 10

Result Overview

In total, the audit team found the following issues.

MONACO PROTOCOL v0.9.0

Issue	Impact	Status
[M-1] Unprocessed delay-expired orders	Medium	Resolved
[L-1] Add Voided to the TRANSFER_SURPLUS_ALLOWED_STATUSES	Low	Resolved
[I-1] inplay not explicitly initialized	Informational	Resolved
[I-2] Use market status to safeguard settle_market_position	Informational	Resolved
[Q-1] Is inplay_order_delay:u8 (255 seconds) large enough?	Question	Resolved
[Q-2] move_market_to_inplay range check	Question	Resolved

Findings in Detail

IMPACT – MEDIUM

[M-1] Unprocessed delay-expired orders

For a delay-expired order (`order.delay_expiration_timestamp <= now`), if it hasn't been processed by `process_delay_expired_orders()`, its liquidity (`order.liquidity_to_add`) has not been added to the pool.

In `cancel_order()`, to skip unexpired orders, it checks the `order.delay_expiration_timestamp`. However, it's possible the order is expired but not processed so its liquidity has not been added to the pool.

Considering invoking `updated_liquidity_with_delay_expired_orders` first or checking the order in the queue to see if it becomes a regular one.

```
/* monaco_protocol/src/instructions/order/cancel_order.rs */
pub fn cancel_order(ctx: Context<CancelOrder>) -> Result<()> {
    let order = &ctx.accounts.order;
    let now = Clock::get().unwrap().unix_timestamp;
    require!(
        !ctx.accounts.market.inplay || order.delay_expiration_timestamp <= now,
        CoreError::InplayDelay
    );
```

Resolution

This issue has been addressed by the commit `fbdf40d`.

IMPACT – LOW

[L-1] Add Voided to the TRANSFER_SURPLUS_ALLOWED_STATUSES

```
/* monaco_protocol/src/instructions/market/escrow.rs */
const TRANSFER_SURPLUS_ALLOWED_STATUSES: [MarketStatus; 2] =
    [MarketStatus::Settled, MarketStatus::ReadyToClose];

pub fn transfer_market_escrow_surplus<'info>(
) -> Result<()> {
    require!(
        TRANSFER_SURPLUS_ALLOWED_STATUSES.contains(&market.market_status),
        CoreError::MarketInvalidStatus
    );
    transfer::transfer_market_escrow_surplus(market_escrow, destination, token_program, market)
}
```

`ready_to_close()` (`Settled/Voided` -> `ReadyToClose`) requires the `market_escrow.amount = 0`. Similar to `Settled`, `Voided` should be added to `TRANSFER_SURPLUS_ALLOWED_STATUSES`. Otherwise, the market state transition will break.

```
/* monaco_protocol/src/instructions/market/update_market_status.rs */
pub fn ready_to_close(market: &mut Market, market_escrow: &TokenAccount) -> Result<()> {
    require!(
        Settled.eq(&market.market_status) || Voided.eq(&market.market_status),
        CoreError::MarketNotSettledOrVoided
    );

    require!(
        market_escrow.amount == 0_u64,
        CoreError::SettlementMarketEscrowNonZero
    );

    market.market_status = ReadyToClose;
    Ok(())
}
```

Resolution

This issue was fixed by the commit `c123f57`.

IMPACT – INFO

[I -1] inplay not explicitly initialized

inplay is not initialized in create() @ src/instructions/market/create_market.rs:012

```
/* monaco_protocol/src/state/market_account.rs */
pub struct Market {
    pub inplay: bool,
```

Resolution

This issue has been fixed by the commit dabab9a

IMPACT – INFO

[I -2] Use market status to safeguard settle_market_position

This is something worth considering. No need to change for now.

Similar to `void_market_position()`, consider guarding the `settle_market_position` explicitly using market status, although non-empty `market_winning_outcome_index` implies the market has at least to be `ReadyForSettlement`. It is easier to read and less error-prone.

```rust
/* monaco_protocol/src/instructions/market_position/settle_market_position.rs */
pub fn settle_market_position(ctx: Context<SettleMarketPosition>) -> Result<()> {
    let market_position = &mut ctx.accounts.market_position;
    if market_position.paid {
        log::sol_log("market position has already been paid out");
        return Ok(());
    }
    let market_account = &ctx.accounts.market;
    // validate the market is settled
    require!(
        market_account.market_winning_outcome_index.is_some(),
        CoreError::SettlementMarketNotSettled
    );

    market_position.paid = true;
```

When market state is Open, it's possible to reset `market_position.paid` by creating new orders.

```rust
/* monaco_protocol/src/instructions/market_position/create_market_position.rs */
pub fn create_market_position(
    purchaser: &Signer,
    market: &Account<Market>,
    market_position: &mut Account<MarketPosition>,
) -> Result<()> {
    market_position.paid = false;
    Ok(())
}
```

Resolution

No action needed at this time.

IMPACT – QUESTION

[Q-1] Is inplay_order_delay:u8 (255 seconds) large enough?

```
/* monaco_protocol/src/state/market_account.rs */
pub struct Market {
    pub inplay_order_delay: u8,
```

Resolution

The team stated that typically this delay is between 3 and 10 seconds and they have internal consensus that a maximum of 255 seconds should suffice for normal usage.

IMPACT – QUESTION

[Q-2] move_market_to_inplay range check

At line 398, `<=` instead of `<`?

```
/* monaco_protocol/src/lib.rs */
386 | pub fn move_market_to_inplay(ctx: Context<UpdateMarketUnauthorized>) -> Result<()> {
396 |     // set it `true` only if now is after event start
397 |     require!(
398 |         market.event_start_timestamp < now,
399 |         CoreError::MarketEventNotStarted,
400 |     );
```

Resolution

This question has been addressed by the commit `e4fa5ee`.

Appendix: 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 the scope of this work

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™