Security Assessment Report

Monaco Protocol v0.10.0

June 23, 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":**
  - v0.10.0, commit `3555b8469a3327b73b89f3b9b34129fd17dc93e9`

The review revealed 3 issues, which have been fixed. The review was finalized and concluded with commit `b1054206aea85a808339aa6109b8388dfbdf1d85`.

Table of Contents

Result Overview

In total, the audit team found the following issues.

MONACO PROTOCOL v0.10.0		
Issue	Impact	Status
[L-1] Negative profit after commission	Low	Resolved
[I-1] market_matching_pool inplay check	Informational	Resolved
[I-2] Typo – inconsistent error type	Informational	Resolved

Findings in Detail

MONACO_PROTOCOL

[L-1] Negative profit after commission

When calculating the total payout amount, the profit after commission can become a negative number. In particular, `position_profit` may be smaller than the two parts of the commissions: `protocol_commission` and `total_product_commission`

In that case, the profit after commission will be smaller than 0. After adding `max_exposure`, the `total_payout` may still be larger than `0` such that the type conversion won't fail.

```
/* instructions/market_position/settle_market_position.rs */
057 | let total_payout = position_profit
058 |     // protocol_commission > 0 only if position_profit > 0
059 |     .checked_sub(i128::from(protocol_commission))
060 |     .ok_or(CoreError::SettlementPaymentCalculation)?
061 |     .checked_sub(i128::from(total_product_commission))
062 |     .ok_or(CoreError::SettlementPaymentCalculation)?
063 |     .checked_add(i128::from(max_exposure))
064 |     .ok_or(CoreError::SettlementPaymentCalculation)?;
```

Test

```
#[test]
fn calculate_commission_payments_one_product_multiple_rates() {
    let protocol_product = Product {
        authority: Default::default(),
        payer: Default::default(),
        commission_escrow: Pubkey::new_unique(),
        product_title: "".to_string(),
        commission_rate: 45.0,
    };
    let market_escrow = Pubkey::new_unique();
    let product_pk = Pubkey::new_unique();
    let matched_risk_for_product = vec![
        ProductMatchedRiskAndRate {
            product: product_pk,
            risk: 5,
            rate: 62.0,
        },
        ProductMatchedRiskAndRate {
```

```
            product: product_pk,
            risk: 5,
            rate: 62.0,
        },
    ];
    let market_position = MarketPosition {
        purchaser: Default::default(),
        market: Default::default(),
        paid: false,
        market_outcome_sums: vec![],
        outcome_max_exposure: vec![],
        payer: Default::default(),
        matched_risk: 10,
        matched_risk_per_product: matched_risk_for_product,
    };
    let position_profit = 100;
    let protocol_commission = calculate_commission(
        protocol_product.commission_rate,
        position_profit,
    );
    let (total_product_commission, payments) = calculate_product_commission_payments(
        protocol_product.commission_rate,
        market_escrow,
        &market_position,
        position_profit,
    );
    println!("position_profit = {}, protocol_commission = {}, total_product_commission = {}",
position_profit, protocol_commission, total_product_commission);
    let after_commission : i128 = position_profit - i128::from(protocol_commission) -
i128::from(total_product_commission);
    println!("after_commission = {}", after_commission);
}
```

Result

```
running 1 test
position_profit = 100, protocol_commission = 45, total_product_commission = 56
after_commission = -1
```

Resolution

This issue has been fixed by commit 9fd2d08. Calculating the remainder using Decimals instead of u64 avoided unnecessary rounding issues.

MONACO_PROTOCOL

[I-1] market_matching_pool inplay check

Orders may still be in the pool if the `inplay` status is not set for `market_matching_pool`.

```
/* instructions/order/cancel_preplay_order_post_event_start.rs */
009 | pub fn cancel_preplay_order_post_event_start(
010 |     ctx: Context<CancelPreplayOrderPostEventStart>,
011 | ) -> Result<()> {
012 |     let order = &ctx.accounts.order;
013 |     let market = &ctx.accounts.market;
014 |
015 |     // market is open + in inplay mode + and cancellation is the intended behaviour
016 |     require!(
017 |         [MarketStatus::Open].contains(&market.market_status),
018 |         CoreError::CancelationMarketStatusInvalid
019 |     );
020 |     require!(market.inplay, CoreError::CancelationMarketNotInplay);
021 |     require!(
022 |         MarketOrderBehaviour::CancelUnmatched.eq(&market.event_start_order_behaviour),
023 |         CoreError::CancelationMarketOrderBehaviourInvalid
024 |     );
025 |
026 |     // order is (open or matched) + created before market event start
027 |     require!(
028 |         [Open, Matched].contains(&order.order_status),
029 |         CoreError::CancelationOrderStatusInvalid
030 |     );
031 |     require!(
032 |         order.creation_timestamp < market.event_start_timestamp,
033 |         CoreError::CancelationOrderCreatedAfterMarketEventStarted
034 |     );
```

Consider adding the pool's `inplay` check in `cancel_preplay_order_post_event_start` after L34.

```
if !market_matching_pool.inplay {
    market_matching_pool.move_to_inplay(&market.event_start_order_behaviour);
}
```

Resolution

This issue has been fixed by commit [3b78d39](3b78d39).

MONACO_PROTOCOL

[I-2] Typo – inconsistent error type

At line 230, `CancelationPurchaserMismatch` should be `CancelationMarketMismatch`.

```
/* MonacoProtocol/programs/monaco_protocol/src/context.rs */
217 | pub struct CancelPreplayOrderPostEventStart<'info> {
230 |     #[account(mut, address = order.market @ CoreError::CancelationPurchaserMismatch)]
231 |     pub market: Box<Account<'info, Market>>,
```

Resolution

This issue has been fixed by commit [c5eef54](c5eef54).

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™