Security Assessment Report

Monaco Protocol v0.13.0

December 26, 2023

Summary

The Sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the Monaco Protocol smart contracts.

The artifact of the audit was the source code of the following programs excluding tests in the repository at https://github.com/MonacoProtocol/protocol.

The initial audit was done on the following versions and revealed 4 issues or questions.

program	type	commit
Monaco Protocol	Solana	`112bda2ab1cba1170734034e83d9364d421be11e`

The post-audit review was done on the following versions to check if the reported issues have been addressed.

program	type	commit
Monaco Protocol	Solana	`409b6e7ea933c9cdd9c88934bcfa3d9e623311d3`

This report describes the findings and resolutions in detail.

Table of Contents

Result Overview

Issue	Impact	Status
MONACO PROTOCOL		
[I-1] Close cancelled orders	Info	Resolved
[I-2] Only settle orders and positions in ReadyForSettlement status	Info	Resolved
[I-3] Check paid status before closing market position accounts	Info	Resolved
[I-4] MatchingQueue is initialized with no capacity	Info	Resolved

Findings in Detail

MONACO PROTOCOL

[I-1] Close cancelled orders

In "`cancel_preplay_order_post_event_start()`", for orders that are never matched (having "`Open`" status or "`voided_stake`" equal to "`stake`"), their status will be set to "`Cancelled`". Additionally, the market's unsettled account counter will decrease by 1.

```
/* programs/monaco_protocol/src/instructions/order/cancel_preplay_order_post_event_start.rs */
011 | pub fn cancel_preplay_order_post_event_start(
016 | ) -> Result<u64> {
046 |     order.void_stake_unmatched(); // <-- void needs to happen before refund calculation
049 |     // if never matched
050 |     if order.stake == order.voided_stake {
051 |         // no more settlement needed
052 |         market.decrement_unsettled_accounts_count()?;
053 |     }

/* programs/monaco_protocol/src/state/order_account.rs */
064 | pub fn void_stake_unmatched(&mut self) {
065 |     self.voided_stake = self.stake_unmatched;
066 |     self.stake_unmatched = 0_u64;
067 |     if self.order_status == OrderStatus::Open {
068 |         self.order_status = OrderStatus::Cancelled;
069 |     }
070 | }
```

The market can then transition to either "`ReadyForSettlement`" or "`ReadyToVoid`". The "`settle_order`" function will not process such cancelled orders, and they will eventually be closed by "`close_order`". A similar process occurs for "`void_order`", except the order status will be set to "`Voided`".

```
/* programs/monaco_protocol/src/instructions/order/cancel_order.rs */
009 | pub fn cancel_order(ctx: Context<CancelOrder>) -> Result<()> {
033 |     ctx.accounts.order.void_stake_unmatched();
045 |     // if never matched close
046 |     if order.stake == order.voided_stake {
047 |         ctx.accounts.market.decrement_account_counts()?;
048 |         ctx.accounts
049 |             .order
050 |             .close(ctx.accounts.purchaser.to_account_info())?;
051 |     }
```

This indicates that these cancelled orders can be closed. In fact, orders cancelled by "`cancel_order`"

will be directly closed.

However, it is also safe to leave them open, given the cancelled order can only be settled or voided once.

Resolution

The team explained that this is a retail feature, meaning it allows users on a product or website to see that their orders have been cancelled, instead of the orders simply disappearing from existence. This issue is resolved.

MONACO PROTOCOL

[I-2] Only settle orders and positions in ReadyForSettlement status

Markets with "ReadyForSettlement" or "Settled" status and some markets with "ReadyToClose" status (those not transited from the "Voided" status) all satisfy the condition at L14 and L24.

```
/* programs/monaco_protocol/src/instructions/order/settle_order.rs */
009 | pub fn settle_order(ctx: Context<SettleOrder>) -> Result<()> {
010 |     let market_account = &mut ctx.accounts.market;
012 |     // validate the market is settled
013 |     require!(
014 |         market_account.market_winning_outcome_index.is_some(),
015 |         CoreError::SettlementMarketNotSettled
016 |     );

/* programs/monaco_protocol/src/instructions/market_position/settle_market_position.rs */
014 | pub fn settle_market_position(ctx: Context<SettleMarketPosition>) -> Result<()> {
021 |     let market_account = &mut ctx.accounts.market;
022 |     // validate the market is settled
023 |     require!(
024 |         market_account.market_winning_outcome_index.is_some(),
025 |         CoreError::SettlementMarketNotSettled
026 |     );
```

However, the order and position settlement instructions should only be allowed when the market is in the "ReadyForSettlement" status:

- The "complete_settlement" instruction, which transits the market from "ReadyForSettlement" to "Settled", requires the "unsettled_accounts_count" is 0. So, all orders and positions should be settled in "ReadyForSettlement" status.
- The "transfer_market_escrow_surplus" instruction is supposed to be invoked after all positions are settled and it's only allowed when the market is in the "Settled" state in the settlement path.

Resolution

The orders and positions can only be settled when the market is in the "ReadyForSettlement" status. This issue has been resovled by commit a8e6bb0.

MONACO PROTOCOL

[I-3] Check paid status before closing market position accounts

In "create_market_position", the "paid" flag is set to "false". In "settle_market_position" and "void_market_position", it is set to "true" such that the position cannot be settled or voided again.

```
/* programs/monaco_protocol/src/instructions/market_position/create_market_position.rs */
006 | pub fn create_market_position(
010 | ) -> Result<()> {
022 |     market_position.paid = false;

/* programs/monaco_protocol/src/instructions/market_position/void_market_position.rs */
009 | pub fn void_market_position(ctx: Context<VoidMarketPosition>) -> Result<()> {
025 |     market_position.paid = true;

/* programs/monaco_protocol/src/instructions/market_position/settle_market_position.rs */
014 | pub fn settle_market_position(ctx: Context<SettleMarketPosition>) -> Result<()> {
068 |     market_position.paid = true;
```

When closing the position accounts, "close_market_position" requires the "ReadyToClose" market status, which implies "market.unsettled_accounts_count" is 0 when the market transits from "ReadyForSettlement" to "Settled".

```
/* programs/monaco_protocol/src/lib.rs */
573 | pub fn close_market_position(ctx: Context<CloseMarketPosition>) -> Result<()> {
574 |     instructions::close::close_market_child_account(&mut ctx.accounts.market)
575 | }

/* programs/monaco_protocol/src/instructions/close.rs */
008 | pub fn close_market_child_account(market: &mut Market) -> Result<()> {
009 |     require!(
010 |         ReadyToClose.eq(&market.market_status),
011 |         CoreError::MarketNotReadyToClose
012 |     );
013 |     market.decrement_unclosed_accounts_count()
014 | }
```

However, the market status check doesn't guarantee the position to be closed has been settled (when the unsettled account counter incorrectly reaches 0). Consider checking the "paid" status of the position before closing.

Resolution

The paid status check was added in the "`close_market_position`" by commit 409b6e7. This issue has been resolved.

MONACO PROTOCOL

[I-4] MatchingQueue is initialized with no capacity

On line 55, "`initialize_matching`" calls "`MatchingQueue::new`" with "`QUEUE_LENGTH`", which is set to 0 (see line 13).

```
/* programs/monaco_protocol/src/instructions/market/update_market_status.rs */
050 | fn intialize_matching_queue(
051 |     matching_queue: &mut MarketMatchingQueue,
052 |     market_pk: &Pubkey,
053 | ) -> Result<()> {
054 |     matching_queue.market = *market_pk;
055 |     matching_queue.matches = MatchingQueue::new(MarketMatchingQueue::QUEUE_LENGTH);
056 |     Ok(())
057 | }

/* programs/monaco_protocol/src/state/market_matching_queue_account.rs */
012 | impl MarketMatchingQueue {
013 |     pub const QUEUE_LENGTH: usize = 0;
014 |
015 |     pub const SIZE: usize = DISCRIMINATOR_SIZE +
016 |         PUB_KEY_SIZE + // market
017 |         MatchingQueue::size_for(MarketMatchingQueue::QUEUE_LENGTH); //matches
018 | }
```

As a result, this matching queue cannot provide any functionality because the vector size is 0.

In addition, "`intialize_matching_queue`" on L50 should be "`initialize_matching_queue`" .

Resolution

The team clarified that this is for an upcoming feature, aiming to reduce the final difference by releasing some components early. The queue won't be in use until the feature is ready to be launched. So for now, it's been sized down to prevent market operators from having to commit more SOL to unnecessary rent exemption fees. This issue is resolved.

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 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.
- 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

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. 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™