Security Assessment Report

Monaco Protocol v0.15.0

August 31, 2024

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Monaco Protocol v0.15.0 smart contracts.

The artifact of the audit was the source code of the following programs, excluding tests, in a private repository.

The initial audit focused on the following versions and revealed 3 issues or questions.

program	type	commit
monaco_protocol	Solana	`aaef66ad1e3ab1f59ab0b94f016793fbe46eb554`

The post-audit review was conducted on the following version to check if the reported issues had been addressed.

program	type	commit
monaco_protocol	solana	`b3b35c061f108f26b192fac175556f3e94de60cd`

This report provides a detailed description of the findings and their respective resolutions.

Table of Contents

Result Overview

Issue	Impact	Status
MONACO_PROTOCOL		
[M-01] Instruction "process_order_match_maker" accepts "taker" orders	Medium	Resolved
[L-01] Validate "update_market_liquidities_with_cross_liquidity" parameters	Low	Resolved
[I-01] Unused outcome "title", "latest_matched_price" and "matched_total"	Info	Resolved

Findings in Detail

MONACO_PROTOCOL

[M-01] Instruction "process_order_match_maker" accepts "taker" orders

The "OrderMatch" records are used to track the "maker" and "taker" of the matched orders.

The only difference between "maker" and "taker" records is that the "maker.pk" is "None", while the "taker.pk" is "Some(pk)".

```
/* monaco_protocol/src/state/market_matching_queue_account.rs */
125 | impl OrderMatch {
132 |     pub fn taker(
133 |         pk: Pubkey,
138 |     ) -> Self {
139 |         OrderMatch {
140 |             pk: Option::Some(pk),
145 |         }
146 |     }
147 |
148 |     pub fn maker(...) -> Self {
149 |         OrderMatch {
150 |             pk: Option::None,
155 |         }
156 |     }
157 | }
```

Instruction "process_order_match_maker" is supposed to process the "maker" record, while the "process_order_match_taker" handles the "taker" records.

In "process_order_match_taker", the handler checks if the "head" of "market_matching_queue.matches" matches the "order". So, the "head" has to be a "taker" record.

However, the "maker" check is not performed in "process_order_match_maker".

```
/* monaco_protocol/src/context.rs */
557 | pub struct ProcessOrderMatchMaker<'info> {
568 |     #[account(
569 |         mut,
570 |         has_one = market @ CoreError::MatchingMarketMismatch,
571 |         constraint = !market_matching_queue.matches.is_empty()
572 |             @ CoreError::MatchingQueueIsEmpty,
573 |     )]
```

```
574 |     pub market_matching_queue: Box<Account<'info, MarketMatchingQueue>>,
575 |     #[account(
576 |         mut,
577 |         has_one = market @ CoreError::MatchingMarketMismatch,
578 |         constraint = maker_market_matching_pool_constraint(&market_matching_queue,
↪  &market_matching_pool)
579 |             @ CoreError::MatchingMarketMatchingPoolMismatch,
580 |     )]
581 |     pub market_matching_pool: Box<Account<'info, MarketMatchingPool>>,
582 |
583 |     #[account(
584 |         mut,
585 |         has_one = market @ CoreError::MatchingMarketMismatch,
586 |         constraint = maker_order_constraint(&market_matching_pool, &order)
587 |             @ CoreError::MatchingPoolHeadMismatch,
588 |     )]
589 |     pub order: Account<'info, Order>,
```

The checks in the context validate that the "market_matching_queue", "market_matching_pool", and "order" are consistent:

"maker_market_matching_pool_constraint" ensures the "head" of "market_matching_queue.matches" shares the same "for_outcome", "market_outcome_index" and "price" with the "market_matching_pool".

"maker_order_constraint" verifies that the "order" is the "head" of the "market_matching_pool.orders" such that the "order" and the "market_matching_pool.orders" are consistent.

However, the handler does not check if the head of the "market_matching_queue.matches" is really for a "marker" by checking if the "order_match.pk" is "None".

As a result, it's possible to pass in a "taker" to the "process_order_match_taker" handler.

Resolution

This issue has been resolved by commit 117ed64a.

MONACO_PROTOCOL

[L-01] Validate "update_market_liquidities_with_cross_liquidity" parameters

The "update_market_liquidities_with_cross_liquidity" is a permissionless instruction and can update the cross matching liquidity. The input parameters should be checked to avoid undesired states.

1. The check against "cross_liquidity" can be bypassed

The user-provided "cross_liquidity" is only used to check if the calculated cross price from "source_liquidities" is the same as "cross_liquidity.price".

However, since "cross_liquidity" and "source_liquidities" are both provided by the caller, this check is not useful.

```
/* monaco_protocol/src/lib.rs */
340 | pub fn update_market_liquidities_with_cross_liquidity(
341 |     ctx: Context<UpdateMarketLiquidities>,
342 |     source_for_outcome: bool,
343 |     source_liquidities: Vec<LiquiditySource>,
344 |     cross_liquidity: LiquiditySource,
345 | ) -> Result<()> {
346 |     instructions::market_liquidities::update_market_liquidities_with_cross_liquidity(
349 |         source_liquidities,
350 |         cross_liquidity,
351 |     )?;
352 |
353 |     Ok(())
354 | }

/*monaco_protocol/src/instructions/market_liquidities/update_market_liquidities_with_cross_liquidity.rs*/
006 | pub fn update_market_liquidities_with_cross_liquidity(
009 |     source_liquidities: Vec<LiquiditySource>,
010 |     cross_liquidity: LiquiditySource,
011 | ) -> Result<()> {
023 |     if let Some(cross_price) = calculate_price_cross(&source_prices) {
024 |         // provided cross_liquidity.price is valid
025 |         if cross_price == cross_liquidity.price {
026 |             if source_for_outcome {
027 |                 market_liquidities.update_cross_liquidity_against(&source_liquidities);
028 |             } else {
029 |                 market_liquidities.update_cross_liquidity_for(&source_liquidities);
030 |             };
031 |         }
032 |     }
035 | }
```

2. Validate the vector length and duplicated outcomes in "source_liquidities"

```
/* monaco_protocol/src/state/market_liquidities.rs */
186 | // recalculates cross liquidity for a given sources
187 | // this method does not validate parameters so value returned might not be real
188 | // assumption is that all (n-1) different sources were passed for the n-outcome market and the
↪  price is of the n-th outcome
189 | pub fn update_cross_liquidity_against(&mut self, sources: &[LiquiditySource]) {
190 |     // silly way of detecting which outcome is supposed to be updated
191 |     // sum of all the outcomes minus sum of all provided ones equals the one we want
192 |     let outcome_count = sources.len().to_u16().unwrap();
193 |     let outcome = (0_u16..=outcome_count).sum::<u16>() - Self::source_outcomes_sum(sources);

/* monaco_protocol/src/state/market_liquidities.rs */
163 | pub fn update_cross_liquidity_for(&mut self, sources: &[LiquiditySource]) {
164 |     // silly way of detecting which outcome is supposed to be updated
165 |     // sum of all the outcomes minus sum of all provided ones equals the one we want
166 |     let outcome_count = sources.len().to_u16().unwrap();
167 |     let outcome = (0_u16..=outcome_count).sum::<u16>() - Self::source_outcomes_sum(sources);
```

Before updating the cross-liquidity, the handler figures out the outcome index to be updated in lines 192-193 above.

This approach works only when the length of "sources" is exactly "market.market_outcomes_count - 1" and there are no items with duplicated outcome index.

However, the handler does not validate these assumptions. Given this instruction is premissionless, considering adding these checks.

3. The "LiquiditySource" elements in "source_liquidities" should be sorted

The current implementation takes the vector of liquidity sources and directly uses it in the comparison functions (e.g. line 203 below).

```
/* monaco_protocol/src/state/market_liquidities.rs */
189 | pub fn update_cross_liquidity_against(&mut self, sources: &[LiquiditySource]) {
199 |     if let Some(cross_price) = calculate_price_cross(&source_prices) {
201 |         Self::set_liquidity(
203 |             Self::sorter_against(outcome, cross_price, sources),
208 |         )
209 |     }
210 | }
```

If the "LiquiditySource" elements in the vector (e.g. "sources") are not sorted, the search func-

tion will treat the same elements with different orders differently, which is undesired in the "update_cross_liquidity_against" scenario.

```
let sources1 = [LiquiditySource::new(1, 2.7), LiquiditySource::new(2, 3.0)];
let sources2 = [LiquiditySource::new(2, 3.0), LiquiditySource::new(1, 2.7)];
```

4. Check duplicated "LiquiditySource" in the "sources"

The handler won't reject a "sources" with duplicated "LiquiditySource" elements.

```
update_market_liquidities_with_cross_liquidity(
    &mut market_liquidities,
    true,
    vec![LiquiditySource::new(0, 2.1), LiquiditySource::new(0, 2.1)],
    LiquiditySource::new(2, 21.0),
)
```

And the cross liquidity inserted looks like the following, which is undesired.

```
[
    MarketOutcomePriceLiquidity {
        outcome: 3,
        price: 21.0,
        sources: [
            LiquiditySource { outcome: 0, price: 2.1 },
            LiquiditySource { outcome: 0, price: 2.1 }
        ],
        liquidity: 10000
    }
]
```

It's better to check the combination of the "outcome" and "price" in a "LiquiditySource" already exists in the liquidity.

Resolution

These issues have been resolved by commit 199b0423.

MONACO_PROTOCOL

[I-01] Unused outcome "title", "latest_matched_price" and "matched_total"

The `"title"`, `"latest_matched_price"` and `"matched_total"` fields in "MarketOutcome" are no longer updated.

```
/* monaco_protocol/src/instructions/market/create_market.rs */
160 | ctx.accounts.outcome.title = title;
161 | ctx.accounts.outcome.latest_matched_price = 0_f64;
162 | ctx.accounts.outcome.matched_total = 0_u64;
```

Resolution

This issue has been resolved by commit e9692be9.

Appendix: Methodology and Scope of Work

Assisted by the Sec3 Scanner developed in-house, the manual audit 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

The Sec3 audit team comprises a group of computer science professors, researchers, and industry veterans with extensive experience in smart contract security, program analysis, testing, and formal verification. We are also building automated 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.