Security Assessment Report

UXD Protocol v3.1.0

June 24th, 2022

Summary

The Sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the UXD Protocol v3.1.0 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. The audit was done on the following commit:

- Commit `6b11a457b46b7cce81270e19ee50cd07cfe9286d`

The audit revealed 6 issues or questions, which were reported to the UXD team. The team responded with the following tag/commit for the post-audit review.

- Tag `v3.1.0` and commit `df527fe9a8e2627ba977c5ee7a5d7b1d717317f2`

The scope of the post-audit review is to validate if the reported issues have been addressed.

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 Soteria 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    - Outdated dependencies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.

CONTRACT UXD PROTOCOL v3.1.0

Issue	Impact	Status
[M-1] Fee rounding issues	Medium	Resolved
[I-1] Consider associated token account checks	Informational	Resolved
[I-2] Design and implementation choices	Informational	Resolved
[I-3] Quote mint upper bound in quote_mint and quote_redeem	Informational	Resolved
[I-4] Quote mint checks	Informational	Resolved
[I-5] Minor: inconsistent or confusing comments	Informational	Resolved

Findings in Detail

IMPACT – MEDIUM

[M-1] Fee rounding issues

`quote_delta` at line 70 can be negative so that the `ceil()` at line 54 may produce a negative number with a smaller absolute value, which seems inconsistent with the purpose of rounding up the fees.

```
/* programs/uxd/src/mango_utils/order_delta.rs */
050 | pub fn taker_fee_amount_ceil(raw_quote_amount: I80F48, taker_fee: I80F48) ->
Result<I80F48> {
051 |     raw_quote_amount
052 |         .checked_mul(taker_fee)
053 |         .ok_or_else(|| error!(UxdError::MathError))?
054 |         .checked_ceil()
055 |         .ok_or_else(|| error!(UxdError::MathError))
056 | }

064 | pub fn derive_order_delta(
065 |     pre_pa: &PerpAccount,
066 |     post_pa: &PerpAccount,
067 |     perp_info: &PerpInfo,
068 | ) -> Result<OrderDelta> {
070 |     let quote_delta = quote_delta(pre_pa, post_pa, perp_info.quote_lot_size)?;
073 |     let fee_delta = taker_fee_amount_ceil(quote_delta, perp_info.effective_fee)?;
074 |
075 |     Ok(OrderDelta {
078 |         fee: fee_delta,
079 |     })
080 | }
```

For example, in `redeem_from_mango_depository`, `quote_delta` is negative.

```
/* repo/programs/uxd/src/instructions/mango_dex/redeem_from_mango_depository.rs */
237 | require!(
238 |     pre_pa.taker_quote >= post_pa.taker_quote,
239 |     UxdError::InvalidOrderDirection
240 | );
241 | let order_delta = derive_order_delta(&pre_pa, &post_pa, &perp_info)?;
```

The following is an example showing the ceil() on negative numbers leads to fewer fees.

```
/* repo/programs/uxd/src/test/mango_utils/test_order_delta.rs */
085 | #[test]
086 | fn test_taker_fee_amount_ceil(
087 |     raw_quote_amount in i64::MIN..i64::MAX,
088 |     taker_fee in 0.0000f64..0.001f64, // 0 bps to 10 bps
089 | ) {
090 |     println!("raw_quote_amount = {}", raw_quote_amount);
091 |     let after_ceil = taker_fee_amount_ceil(I80F48::from_num(raw_quote_amount),
I80F48::from_num(taker_fee))?;
092 |     println!("fee_after_ceil = {}", after_ceil);
096 |     }
097 | }
* expected. rounding up the fees
-------------------------------------------------
raw_quote_amount = 687432926924899829
fee_before_ceil  = 325334881156659.088065821747012
fee_after_ceil   = 325334881156660

* inconsistent fee rounding behavior
-------------------------------------------------
raw_quote_amount = -6924949546447067303
fee_before_ceil  = -4222859722648493.0802370853312
fee_after_ceil   = -4222859722648493  // should be -4222859722648494
```

Resolution

The sign has been considered to round up the absolute value of the fee amount. This issue has been resolved.

IMPACT – INFO

[I-1] Consider associated token account checks

The current mint/owner checks are safe as the owner's signatures are required.

The associated token account checks may be useful too. The ATA check is more strict as it only allows one account, while more than one account can be accepted by the mint/owner check.

```
#[account(
    mut,
    constraint = user_quote.mint == depository.load()?.quote_mint,
    constraint = user_quote.owner == *user.key,
)]
pub user_quote: Box<Account<'info, TokenAccount>>,
// ------------
#[account(
    mut,
    address = get_associated_token_address(user.key, &depository.load()?.quote_mint)
)]
pub user_quote: Box<Account<'info, TokenAccount>>,
```

Resolution

The team has decided not to enforce the ATA checks for the flexibility. It's still safe. This issue has been resolved.

IMPACT – INFO

[I-2] Design and implementation choices

1. The `PerpAccount.quote_position` is not included when calculating the quote delta. This is different from the base delta calculation where the `base_position` is considered. Is this because the `quote_position` won't change in this scenario as the perp orders do not expire?

2. The handler of instruction `mint_with_mango_depository` checks if the perp order is fully fulfilled but `redeem_from_mango_depository` doesn't. Is the check also needed?

```
/* programs/uxd/src/instructions/mango_dex/mint_with_mango_depository.rs */
235 | check_perp_order_fully_filled(max_base_quantity_num, &pre_pa, &post_pa)?;
```

3. The fee accounting operations use both `wrapping_add` and `checked_add`. However, other accounts such as redeemable_circulating_supply, collateral_amount_deposited only use `checked_add`. It's unclear why `wrapping_add` is chosen for some fee operations.

```
/* programs/uxd/src/instructions/mango_dex/mint_with_mango_depository.rs */
417 | // Update the accounting in the Depository and Controller Accounts to reflect changes
418 | fn update_onchain_accounting(
423 | ) -> Result<()> {
435 |     depository.total_amount_paid_taker_fee = depository
436 |         .total_amount_paid_taker_fee
437 |         .wrapping_add(fee_amount);
444 | }

/* programs/uxd/src/instructions/mango_dex/quote_mint_with_mango_depository.rs */
298 | fn update_onchain_accounting(
302 | ) -> Result<()> {
317 |     depository.total_quote_mint_and_redeem_fees = depository
318 |         .total_quote_mint_and_redeem_fees
319 |         .checked_add(fees_accrued.into())
320 |         .ok_or_else(|| error!(UxdError::MathError))?;
327 | }
```

Resolution

The behaviors in issues 1 and 2 are intended. The `checked_add` is used for fee accounting. These issues have been resolved.

IMPACT – INFO

[I-3] Quote mint upper bound in quote_mint and quote_redeem

1. Previous redeemable minted with the quote mint (depository.net_quote_minted) has already been captured by depository.redeemable_amount_under_management (which is used to compute perp_unrealized_pnl), it's not clear why the upper bound needs to consider depository.net_quote_minted again.
2. The comment is confusing. quote_mintable is a u64 integer, while the comment above says it will become negative.

```
/* programs/uxd/src/instructions/mango_dex/quote_mint_with_mango_depository.rs */
185 | // Get how much redeemable has already been minted with the quote mint
186 | let quote_minted = depository.net_quote_minted;
188 | // Only allow quote minting if PnL is negative
189 | require!(
190 |     perp_unrealized_pnl.is_negative(),
191 |     UxdError::InvalidPnlPolarity
192 | );
194 | // Will become negative if more has been minted than the current negative PnL
195 | let quote_mintable: u64 = perp_unrealized_pnl
196 |     .checked_sub(
197 |         I80F48::checked_from_num(quote_minted).ok_or_else(...)?,
198 |     )
199 |     .ok_or_else(|| error!(UxdError::MathError))?
200 |     .checked_abs()
201 |     .ok_or_else(|| error!(UxdError::MathError))?
202 |     .checked_to_num::<u64>()
203 |     .ok_or_else(|| error!(UxdError::MathError))?;
```

quote_redeem_from_mango_depository does the same thing. Besides, the abs() at line 209 seems redundant as the check in 204 makes sure it's positive.

```
/* programs/uxd/src/instructions/mango_dex/quote_redeem_from_mango_depository.rs */
199 | // Get how much redeemable has already been minted with the quote mint
200 | let quote_minted = depository.net_quote_minted;
201 |
202 | // Only allow quote redeeming if PnL is positive
203 | require!(
204 |     perp_unrealized_pnl.is_positive(),
```

```
205 |     UxdError::InvalidPnlPolarity
206 | );
207 |
208 | let quote_redeemable: u64 = perp_unrealized_pnl
209 |     .checked_abs()
210 |     .ok_or_else(|| error!(UxdError::MathError))?
211 |     .checked_to_num::<i128>()
212 |     .ok_or_else(|| error!(UxdError::MathError))?
213 |     .checked_add(quote_minted)
214 |     .ok_or_else(|| error!(UxdError::MathError))?
215 |     .try_into()
216 |     .unwrap();
```

Resolution

The implementation has been refactored. These issues have been resolved.

IMPACT – INFO

[I-4] Quote mint checks

Instruction `quote_mint_with_mango_depository` and `quote_redeem_from_mango_depositor` do not explicitly check if the `quote_mint` is consistent with the quote currency of the mango market such as `mango_group.tokens[QUOTE_INDEX].mint == *quote_mint_key`.

Although the Mango doc mentioned USDT may be a quote currency as well, the current implementation is still safe because of the constraints on the `user_quote` token account and the implicit constraints enforced by the token transfer.

Resolution

Additional `quote_mint` validations have been added. This issue has been resolved.

IMPACT – INFO

[I-5] Minor: inconsistent or confusing comments

1. 0.0004 = 4bps

```
/* programs/uxd/src/mango_utils/perp_info.rs */
020 | // taker_fee : 0.004 = 4bps on most perp markets (as of 02/20/2022)
```

2. A reviewer was confused by the mango_depositories_redeemable_soft_cap comments.

   This cap is actually per mint/redeem operation.

```
/* programs/uxd/src/state/controller.rs */
031 | // The max amount of Redeemable affected by Mint and Redeem operations on
`MangoDepository` instances, variable
032 | //  in redeemable Redeemable Native Amount
033 | pub mango_depositories_redeemable_soft_cap: u64,
```

3. Flag minting_disabled controls redeemable minting via depositing collateral only.

```
/* programs/uxd/src/state/mango_depository.rs */
079 | // Flag for enabling / disabling minting with this depository's collateral_mint
080 | pub minting_disabled: bool,
```

Resolution

These issues have been resolved.

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Sec3 (the "Company") and Soteria FZCO (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.