SOTERIA

Security Assessment Report

UXD Protocol v3.0.1

March 30th, 2022

Summary

The Soteria team was engaged to do a thorough security analysis of the UXD Protocol v3.0.1 Solana smart contract program. The artifact of the audit was the source code of the following on-chain smart contract excluding tests in a private repository:

- Tag `v3.0.1`
- commit `958d36a20a77c0ae1945375ebda7a8a660999f46`

The audit revealed 5 issues including 1 critical vulnerability, which were reported to the UXD team.

The UXD team responded promptly. Several PRs and a merged version were provided for the post-audit review. The scope of the post-audit review is to validate if the reported issues have been addressed. The audit was finalized based on the following version:

- Tag `v3.0.2`
- commit `e95bd2192b23926e1f35e8e784d5ca62e6168294`

This report describes the findings and resolutions in detail.

Table of Contents

Methodology and Scope of Work

Soteria's 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    - Outdated dependencies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

Issue	Impact	Status
[C-1] manipulate perpetual trade parameters and steal assets	Critical	Resolved
[I-1] redundant passthrough accounts/transfers	Informational	Resolved
[I-2] deprecated mango account initialization method	Informational	Accepted
[I-3] unhandled corner scenario in mint_with_mango_depository	Informational	Resolved
[I-4] design choices, clarifications and questions	Informational	Resolved

Findings in Detail

IMPACT - CRITICAL

[C-1] manipulate perpetual trade parameters and steal assets

When calling the following three unprivileged instructions

- mint_with_mango_depository
- redeem_from_mango_depository
- rebalance_mango_depository_lite

the following four accounts should be provided to indicate which perpetual market the users are dealing with.

```
/* programs/uxd/src/instructions/mango_dex/mint_with_mango_depository.rs */
pub struct MintWithMangoDepository<'info> {
    /// #16 [MangoMarkets CPI] `depository`'s `collateral_mint` perp market
    #[account(mut)]
    pub mango_perp_market: AccountInfo<'info>,
    /// #17 [MangoMarkets CPI] `depository`'s `collateral_mint` perp market orderbook bids
    #[account(mut)]
    pub mango_bids: AccountInfo<'info>,
    /// #18 [MangoMarkets CPI] `depository`'s `collateral_mint` perp market orderbook asks
    #[account(mut)]
    pub mango_asks: AccountInfo<'info>,
    /// #19 [MangoMarkets CPI] `depository`'s `collateral_mint` perp market event queue
    #[account(mut)]
    pub mango_event_queue: AccountInfo<'info>,
}
```

However, the existing checks on the mango_perp_market account are insufficient. It's possible to provide valid but inconsistent perp market accounts such that normal users can manipulate the perp order parameters and steal assets.

For example, when users interact with the ETH depository, to be consistent, the PERP-ETH mango perp market accounts are expected. However, as shown below, it's possible to use PERP-BTC or PERP-SOL related accounts and manipulate the transaction parameters.

In particular, through instruction mint_with_mango_depository, attackers may obtain 13x more assets from the contract than the collateral they deposited. In our experiment, this attack can be repeated multiple times.

Details

We use mint_with_mango_depository as an example to show why the existing checks are insufficient and how they can be exploited.

At mint_with_mango_depository.rs:191, the perp market information is loaded using the user provided accounts including mango_perp_market. Such information is then used to compute the price and quantity for the perp market order.

```
/* programs/uxd/src/instructions/mango_dex/mint_with_mango_depository.rs */
175 | pub fn handler(
177 |     collateral_amount: u64,
179 | ) -> UxdResult {
191 |    let perp_info = ctx.accounts.perpetual_info()?;
246 |    mango_program::place_perp_order(
247 |        ctx.accounts
248 |            .into_open_mango_short_perp_context()
250 |        perp_order.price,    // affected by perp_info.price
251 |        perp_order.quantity, // affected by perp_info.base_lot_size
256 |    )?;

366 | pub fn into_open_mango_short_perp_context(
368 | ) -> CpiContext<'_, '_, '_, 'info, mango_program::PlacePerpOrder<'info>> {
369 |     let cpi_accounts = mango_program::PlacePerpOrder {
374 |         mango_perp_market: self.mango_perp_market.to_account_info(),
375 |         mango_bids: self.mango_bids.to_account_info(),
376 |         mango_asks: self.mango_asks.to_account_info(),
377 |         mango_event_queue: self.mango_event_queue.to_account_info(),
378 |     };
381 | }

409 | fn perpetual_info(&self) -> UxdResult<PerpInfo> {
410 |    let perp_info = PerpInfo::new(
414 |        self.mango_perp_market.key,
417 |    )?;
419 |    Ok(perp_info)
420 | }
```

At perp_info.rs:45, the perp_market_key is compared against the registered perp markets in mango_group. In other words, as long as it's a valid perp market key, the perp market and its metadata such as price (perp_info.rs:66-72) will be loaded without considering if the selected SOL/BTC/ETH perp market matches the depository the users are interacting with. As a result, attackers can use inconsistent trade parameters such as price for profit.

```
/* programs/uxd/src/mango_utils/perp_info.rs */
031 | impl PerpInfo {
033 |     pub fn new(
037 |         perp_market_key: &Pubkey,
040 |     ) -> UxdResult<Self> {
             // omit checks on mango_group, mango_cache, mango_account
044 |         let perp_market_index = mango_group
045 |             .find_perp_market_index(perp_market_key)
049 |         PerpInfo::init(
050 |             &mango_group,
051 |             &mango_account,
052 |             &mango_cache,
053 |             perp_market_index,
054 |         )
055 |    }

057 |    pub fn init(
058 |        mango_group: &MangoGroup,
059 |        mango_account: &MangoAccount,
060 |        mango_cache: &MangoCache,
061 |        perp_market_index: usize,
062 |    ) -> UxdResult<Self> {
065 |        Ok(PerpInfo {
066 |            market_index: perp_market_index,
067 |            price: mango_cache.price_cache[perp_market_index].price,
068 |            base_lot_size: I80F48::from_num(
069 |                mango_group.perp_markets[perp_market_index].base_lot_size,
071 |            quote_lot_size: I80F48::from_num(
072 |                mango_group.perp_markets[perp_market_index].quote_lot_size,
```

Next, let's check what additional constraints should be satisfied. mango::place_perp_order requires the mango_bids, mango_asks and mango_event_queue accounts match the corresponding fields in mango_perp_market. In other words, it should be fine if the provided accounts are valid and about the same perp market.

```
/* https://github.com/blockworks-foundation/mango-v3/blob/v3.4.0/program/src/processor.rs#L2363 */
fn place_perp_order(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
) -> MangoResult {
    let (fixed_ais, open_orders_ais, opt_ais) =
        array_refs![accounts, NUM_FIXED, MAX_PAIRS; ..;];
    let [
        perp_market_ai,     // write
        bids_ai,            // write
        asks_ai,            // write
        event_queue_ai,     // write
    ] = fixed_ais;
    let mut perp_market =
    //  perp_market_ai.owner == program_id
        PerpMarket::load_mut_checked(perp_market_ai, program_id, mango_group_ai.key)?;
    // bids_ai.key == perp_market.bids, asks_ai.key == perp_market.asks
    let mut book = Book::load_checked(program_id, bids_ai, asks_ai, &perp_market)?;
    let mut event_queue =
    // event_queue.key == perp_market.event_queue
        EventQueue::load_mut_checked(event_queue_ai, program_id, &perp_market)?;
```

PoC on the Devnet

To show the issue, atop the provided e2e tests, we created the PoCs, chose the ETH depository and explored perp markets based on the token price relations:

- PERP-ETH, where the perp market matches the depository
- PERP-SOL, where the SOL price is lower than ETH
- PERP-BTC, where the BTC price is higher than ETH

The results together with the Devnet transaction URLs are listed as follows:

(1) ETH depository and PERP-ETH

```
[XXX amount 0.0334664730028807 ETH]
mintWithMangoDepositoryTest
  [mint XXX] depository.collateralMintSymbol = ETH, perp_symbol = ETH
  [mint XXX] mango_account_pda = CxSPfpjQ1Hq6TwLmQiYZQD3RFYAbACcE9xrSXjMerJ7o
  perp price is 2988.5385 USDC
https://explorer.solana.com/tx/59rdmWbwhr9YW7ii8yeE8EiH68VE12hxu4PMxy9WG6brEPVNyaRJfQuh7VKZdr8PDv8B2LoELvrKKGTfGkJmzgEK?cluster=devnet
  Efficiency 99.52 %
  Minted 98.149 UXD by locking 0.033 ETH (+~ takerFees = 0.049312 UXD , +~ slippage = 0.424872 UXD )
```

In this normal transaction, 98.149 UXD was minted for the deposited 0.033 ETH or $98.62, which is reasonable.

(2) ETH depository and PERP-SOL market

```
[XXX amount 33.51826701978831 ETH]
mintWithMangoDepositoryTest
  [mint XXX] depository.collateralMintSymbol = ETH, perp_symbol = SOL
  [mint XXX] mango_account_pda = CxSPfpjQ1Hq6TwLmQiYZQD3RFYAbACcE9xrSXjMerJ7o
  perp price is 2983.4511 USDC
https://explorer.solana.com/tx/44E75gqDk1i8CD8DBH6RHeJgDjmHW1Ac68jXtURzspBPK6LCq1ckXYWdkSnQGnLk5uLjJU
fGmTfM6MRutNDBmJSs?cluster=devnet
  Efficiency 0 %
  Minted 2.710315 UXD by locking 30 ETH (+~takerFees = 44.751767 UXD, +~slippage = 89456.070978 UXD )
```

In this case, we used the PERP-SOL parameters for the ETH depository. After depositing 30 ETH or $89503.53, only 2.710315 UXD was minted, which is not helpful for attackers. Note: comparing to the other two scenarios, due to the larger base lot size of SOL, a larger amount of the deposit assets was used to satisfy the positive quantity requirement when placing the perp order.

(3) ETH depository and PERP-BTC

```
[XXX amount 0.03344953143060134 ETH]
mintWithMangoDepositoryTest
  [mint XXX] depository.collateralMintSymbol = ETH, perp_symbol = BTC
  [mint XXX] mango_account_pda = CxSPfpjQ1Hq6TwLmQiYZQD3RFYAbACcE9xrSXjMerJ7o
  perp price is 2988.8975 USDC
https://explorer.solana.com/tx/5cXJSNN3sL9n46wNjRj3UxMvDEuFSBwDNB2mfJmwY7vhV7SJAwBoQdb8doJ1rMK6dPbgX1
BbzFDpmPxfZeUVaKR2?cluster=devnet
  Efficiency 1407.21 %
  Minted 1404.785955 UXD by locking 0.0334 ETH (+~takerFees=0.049914 UXD, +~slippage=1304.908265 UXD)
```

This is the happy path for attackers. we used the PERP-BTC parameters for the ETH depository. 1404.785955 UXD was issued for the deposited 0.0334 ETH or $99.83, where the return was 14x times of the deposit. More importantly, we can repeatedly do this to get more UXD.

Resolution

Since the UXD protocol has already been deployed to the Mainnet before the audit, we immediately reported this issue to the UXD team. The UXD team promptly confirmed and fixed this issue.

IMPACT – INFO

[I-1] redundant passthrough accounts/transfers

The comments/README in the UXD implementation indicate that the limits on the number of accounts allowed per instruction and the computation costs are challenging problems. We also observed the current computation unit usages are usually high. We easily hit the limit and had to rebuild/redeploy when we run the end-to-end tests on the Devnet.

We noticed that every deposit/withdrawal has a passthrough account. According to the comment, the reason to have the passthrough accounts is “MangoAccounts can only transact with the TAs owned by their authority”, which seems not the case in the current implementation. We found the passthrough accounts may be simplified or removed to save the computation costs.

Deposit passthrough accounts/transfers

```
/* programs/uxd/src/instructions/mango_dex/deposit_insurance_to_mango_depository.rs */
129 | // - Transfers insurance to the passthrough account
130 | token::transfer(
131 |    ctx.accounts.into_transfer_to_passthrough_context(),
132 |    insurance_amount,
133 | )?;
135 | // - Deposit Insurance to Mango Account
136 | mango_program::deposit(
137 |    ctx.accounts
138 |        .into_deposit_to_mango_context()
139 |        .with_signer(depository_signer_seeds),
140 |    insurance_amount,
141 | )?;

159 | pub fn into_transfer_to_passthrough_context(
160 |    &self,
161 | ) -> CpiContext<'_, '_, '_, 'info, token::Transfer<'info>> {
162 |     let cpi_accounts = Transfer {
163 |        from: self.authority_insurance.to_account_info(),
164 |        to: self
165 |            .depository_insurance_passthrough_account
167 |        authority: self.authority.to_account_info(),
168 |     };

173 | pub fn into_deposit_to_mango_context(
    &self,
) -> CpiContext<'_, '_, '_, 'info, mango_program::Deposit<'info>> {
    let cpi_accounts = mango_program::Deposit {
        owner: self.depository.to_account_info(),
        mango_vault: self.mango_vault.to_account_info(),
        token_program: self.token_program.to_account_info(),
        owner_token_account: self
            .depository_insurance_passthrough_account
    };
```

For example, in `deposit_insurance_to_mango_depository.rs`, there are two transfers:

1. `authority_insurance` to `depository_insurance_passthrough_account` and
2. `depository_insurance_passthrough_account` to `mango_vault`.

The `owner` of `deposiitory_insurance_passthrough_account` is `token_program`. And, its `authority` is `depository`. In addition, the `owner` of `depository` is the UXD contract.

So, their `authority` or `owner` accounts are unrelated to Mango.

```
/* https://github.com/blockworks-foundation/mango-v3/blob/v3.4.0/program/src/processor.rs#L879 */
fn deposit(program_id: &Pubkey, accounts: &[AccountInfo], quantity: u64) -> MangoResult<()> {
    let accounts = array_ref![accounts, 0, NUM_FIXED];
    let [
        owner_ai,               // read
        vault_ai,               // write
        token_prog_ai,          // read
        owner_token_account_ai, // write
    ] = accounts;
        // Note: a check for &mango_account.owner == owner_ai.key doesn't exist on purpose
    invoke_transfer(token_prog_ai, owner_token_account_ai, vault_ai, owner_ai, &[], quantity)?;

/* https://github.com/blockworks-foundation/mango-v3/blob/v3.4.0/program/src/processor.rs#L6628 */
fn invoke_transfer<'a>(
    token_prog_ai: &AccountInfo<'a>,
    source_ai: &AccountInfo<'a>,
    dest_ai: &AccountInfo<'a>,
    authority_ai: &AccountInfo<'a>,
    signers_seeds: &[&[&[u8]]],
    quantity: u64,
) -> ProgramResult {
    let transfer_instruction = spl_token::instruction::transfer(
        &spl_token::ID,
        source_ai.key,
        dest_ai.clone(),
        authority_ai.key,
    )?;
```

However, if we look at the constraints in mango_program::deposit, the only requirement is that owner_ai owns owner_token_account.

Therefore, the passthrough account and the transfer can be eliminated by directly depositing from authority_insurance and set the owner to self.authority.

```
pub fn into_deposit_to_mango_context(
    &self,
) -> CpiContext<'_, '_, '_, 'info, mango_program::Deposit<'info>> {
    let cpi_accounts = mango_program::Deposit {
        owner:self.authority.to_account_info(),
        owner_token_account: self.authority_insurance.to_account_info(),
```

Transaction details of the deposit_insurance_to_mango_depository instruction on the Devnet before and after removing deposit passthrough account/transfer are shown in:

- The transaction before. It consumed 59151 computation units.
- The transaction after. It consumed 53665 computation units

Withdraw passthrough accounts/transfers

Similarly, the passthrough accounts/transfers for `mango_program::withdraw` can be simplified too.

```
/* programs/uxd/src/instructions/mango_dex/redeem_from_mango_depository.rs */
mango_program::withdraw(
    ctx.accounts
       .into_withdraw_collateral_from_mango_context()
    order_delta.collateral,
)?;

token::transfer(
    ctx.accounts
       .into_transfer_collateral_to_user_context()
    order_delta.collateral,
)?;

pub fn into_withdraw_collateral_from_mango_context(
) -> CpiContext<'_, '_, '_, 'info, mango_program::Withdraw<'info>> {
    let cpi_accounts = mango_program::Withdraw {
        mango_account: self.depository_mango_account.to_account_info(),
357 |         owner: self.depository.to_account_info(),
362 |         token_account: self
363 |             .depository_collateral_passthrough_account
367 |     };
370 | }

372 | pub fn into_transfer_collateral_to_user_context(
374 | ) -> CpiContext<'_, '_, '_, 'info, token::Transfer<'info>> {
376 |     let cpi_accounts = token::Transfer {
377 |         from: self
378 |             .depository_collateral_passthrough_account
380 |         to: self.user_collateral.to_account_info(),
381 |         authority: self.depository.to_account_info(),
382 |     };
384 | }
```

In fact, in mango_program::withdraw,

account token_account at redeem_from_mango_depository.rs:362 or token_account_ai at processor.rs:1271 is directly passed to the SPL token transfer instruction.

```
/* https://github.com/blockworks-foundation/mango-v3/blob/v3.4.0/program/src/processor.rs#L1269 */
1252| fn withdraw(
1253|     program_id: &Pubkey,
1254|     accounts: &[AccountInfo],
1255|     quantity: u64,
1256|     allow_borrow: bool,
1257| ) -> MangoResult<()> {
1261|     let [
1268|         vault_ai,           // write
1269|         token_account_ai,   // write
1270|         signer_ai,          // read
1271|         token_prog_ai,      // read
1272|     ] = fixed_ais;
1329|     invoke_transfer(
1330|         token_prog_ai,
1331|         vault_ai,
1332|         token_account_ai,
1333|         signer_ai,
1336|     )?;
```

So, there is no additional constraints and the transfer to the user can be done directly.

```
pub fn into_withdraw_collateral_from_mango_context(
) -> CpiContext<'_, '_, '_, 'info, mango_program::Withdraw<'info>> {
    let cpi_accounts = mango_program::Withdraw {
        token_account: self.user_collateral.to_account_info(),
    };
}
```

Transaction details of redeem_from_mango_depository on the Devnet before and after removing withdraw passthrough account/transfer are shown in:

- The transaction before simplifications. It consumed 165,459 computation units.
- The transaction after simplifications. It consumed 159,994 computation units.

Resolution

The UXD team acknowledged the findings and removed all passthrough accounts in relevant instructions.

IMPACT - INFO

[I-2] deprecated mango account initialization method

MangoInstruction::InitMangoAccount is deprecated. Accounts created with this function cannot be closed without upgrading with UpgradeMangoAccountV0V1, which seems not what instruction migrate_mango_depository_to_v2 does.

```
/* programs/uxd/src/mango_program/init_mango_account.rs */
fn initialize_mango_account_instruction(
) -> Result<Instruction, ProgramError> {
    let data = mango::instruction::MangoInstruction::InitMangoAccount.pack();

pub fn initialize_mango_account<'info>(ngoAccount<'info>>,
) -> ProgramResult {
    let ix = initialize_mango_account_instruction(
    solana_program::program::invoke_signed(
        &ix,
}

/* programs/uxd/src/instructions/register_mango_depository.rs */
pub fn handler(ctx: Context<RegisterMangoDepository>) -> UxdResult {
    mango_program::initialize_mango_account(
    )?;
```

Resolution

The UXD team acknowledged the finding. The team already knew the deprecated Mango account initialization method issue.

The team stated that they will update the method in the future because there is no necessity to do so at this moment.

IMPACT – INFO

[I-3] unhandled corner scenario in mint_with_mango_depository

mint_with_mango_depository checks collateral_amount and makes sure it's larger than 0. However, it's possible collateral_amount < perp_info.base_lot_size. When it happens, the quantity for mango_program::place_perp_order is 0 and triggers MangoErrorCode::InvalidParam.

```
/* programs/uxd/src/instructions/mango_dex/mint_with_mango_depository.rs */
let base_lot_amount = I80F48::from_num(collateral_amount)
   .checked_div(perp_info.base_lot_size)
   .ok_or(math_err!())?
   .floor();

let perp_order = Order {
   quantity: base_lot_amount.checked_to_num().ok_or(math_err!())?,

mango_program::place_perp_order(
   perp_order.quantity,
)?;


/* https://github.com/blockworks-foundation/mango-v3/blob/v3.4.0/program/src/processor.rs#L2374 */
fn place_perp_order(
   quantity: i64,
) -> MangoResult {
   check!(quantity > 0, MangoErrorCode::InvalidParam)?;
```

Resolution

The UXD team acknowledged the finding and added checks for such cases.

IMPACT – INFO

[I-4] design choices, clarifications and questions

- There are several privileged instructions. However, there is no owner/admin transfer instruction. We were wondering what would be the plan if the owner's credential is lost. Depending on the needs, Multisig might be useful too.

- Maybe it's a good idea to merge `set_redeemable_global_supply_cap` and `set_mango_depositories_redeemable_soft_cap` such that it's easier to validate two caps and keep the correct invariant.

- `Total_amount_paid_taker_fee` is computed but not used in the contract. Is there a different component consuming this information?

- `I80F48` multiplication operations are expensive. Avoid when possible.

Resolution

The UXD team acknowledged the findings. They are either intended by design or have been addressed in the newer versions that are not in the scope of this audit.

DISCLAIMER

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