SOTERIA

Security Assessment Report

Mean Protocol

March 31st, 2022

Summary

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

- DDCA Protocol
  - branch `develop`
  - commit `7c4d7899c40f577664cff8619d34ccea6baac23e`
  - path `mean-core/ddca/programs/ddca/`
- Hybrid Liquidity Aggregator (HLA) Protocol
  - branch `develop`
  - commit `7c4d7899c40f577664cff8619d34ccea6baac23e`
  - path `mean-core/hybrid-liquidity-ag/programs/hla/`

The audit revealed 9 issues, which were reported to the Mean DAO team.

The Mean DAO team responded and provided the DDCA/HLA versions 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:

- DDCA Protocol
  - branch `main`
  - commit `637b2b12c3859a89b3c82b96c92e10489aad7387`
  - path `mean-core/ddca/programs/ddca/`
- Hybrid Liquidity Aggregator (HLA) Protocol
  - branch `main`
  - commit `637b2b12c3859a89b3c82b96c92e10489aad7387`
  - path `mean-core/hybrid-liquidity-ag/programs/hla/`

This report describes the findings 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. The client did not provide the formal specifications but provided documentations.

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    - Outdated dependencies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 DDCA

Issue	Impact	Status
[DDCA-L-1] confusing refund computation flow	Low	Resolved
[DDCA-L-2] integer overflows	Low	Resolved
[DDCA-L-3] logic error when updating swap_avg_rate	Low	Resolved
[DDCA-I-1] partial account validation	Informational	Resolved
[DDCA-I-2] unused variable swap_slippage	Informational	Resolved
[DDCA-I-3] repeated statements	Informational	Resolved
[DDCA-I-4] function create does not update laste_deposit_ts	Informational	Resolved

Contract HLA

Issue	Impact	Status
[HLA-I-1] account validations	Informational	Resolved
[HLA-I-2] dependencies versioning issues	Informational	Resolved

Findings in Detail

Contract DDCA

IMPACT - LOW

[DDCA-L-1] confusing refund computation flow

When emptying the token accounts before closing the DDCA account, it seems the logic is to split the balance in ddca_from_token_account into two parts:

- fees (from_withdraw_fee), which goes to the operating account
- balance after paying the fees (from_token_amount), which will be refunded.

Although the current refund amount (from_token_amount) computation is correct, it's confusing as it uses the implicit knowledge that the account data is not automatically synced with the underlying storage. It may be inconsistent if this behavior changes.

- **Step 1**: from_withdraw_fee is transferred from ddca_from_token_account in lines 435-441. In the underlying storage, from_withdraw_fee will be deducted from the account. However, it's not deducted from ddca_from_token_account.amount in the program, which may cause confusion and inconsistency.
- **Step 2**: if the ddca_from_token_account balance is larger, in lines 448-450, the amount to be refunded is ddca_from_token_account - from_withdraw_fee. The amount is correct because ddca_from_token_account in program is not affected by the transfer in step 1.
- **Step 3**: refund and transfer the remaining balance from ddca_from_token_account to owner_from_token_account.

```
/* ddca/programs/ddca/src/lib.rs */
435 | if from_withdraw_fee > 0 {
436 |     token::transfer(
            // [from] ddca_from_token_account  [to] operating_from_token_account
437 |         ctx.accounts.into_transfer_from_fee_to_operating_context()
438 |         .with_signer(&[&seeds[..]]),
439 |         from_withdraw_fee,
440 |     )?;
}
let from_token_amount = {
    if from_withdraw_fee > ctx.accounts.ddca_from_token_account.amount {
        ctx.accounts.ddca_from_token_account.amount
    }
    else {
        ctx.accounts.ddca_from_token_account.amount
        .checked_sub(from_withdraw_fee)
        .ok_or(ProgramError::InvalidArgument)?
    }
};
if from_token_amount > 0 {
    token::transfer(
        // [from] ddca_from_token_account   [to] owner_from_token_account
        ctx.accounts
            .into_transfer_from_to_owner_context()
            .with_signer(&[&seeds[..]]),
            from_token_amount,
    )?;
}
```

Similarly, the process of emptying the ddca_to_token_account account (ddca/src/lib.rs:475-502) has the same issue.

Resolution

The team acknowledged the finding and made the changes: Fees and refund amounts are explicitly computed before the transfers.

IMPACT – LOW

[DDCA-L-2] integer overflows

The following snippet has several unchecked version of the arithmetic operators.

```
/* ddca/programs/ddca/src/lib.rs */
157 | pub fn wake_and_swap<'info>(
158 |  ctx: Context<'_, '_, '_, 'info, WakeAndSwapInputAccounts<'info>>,
161 | ) -> ProgramResult {
165 |   let interval = ctx.accounts.ddca_account.interval_in_seconds;
169 |   let max_delta_in_secs = cmp::max(cmp::min(interval / 20, 7200), 1);
171 |   let max_delta_in_secs = cmp::max(cmp::min(interval / 20, 7200), 300);
172 |   let prev_checkpoint = (now_ts - start_ts) / interval;
173 |   let prev_ts = start_ts + prev_checkpoint * interval;
174 |   let next_checkpoint = prev_checkpoint + 1;
175 |   let next_ts = start_ts + next_checkpoint * interval;
182 |   if now_ts >= (prev_ts - max_delta_in_secs) && now_ts <= (prev_ts + max_delta_in_secs) {
184 |   }
185 |   else if now_ts >= (next_ts - max_delta_in_secs) && now_ts <= (next_ts + max_delta_in_secs) {
187 |   }
237 |   let swap_rate =  u64::try_from(
238 |       (to_amount_delta as u128)
239 |       .checked_mul(10u128.pow(ctx.accounts.ddca_account.from_mint_decimals.into()))).unwrap()
289 | }

795 | #[account]
796 | pub struct DdcaAccount {
809 |    pub interval_in_seconds: u64, //8 bytes
819 | }
```

In particular, the value of variable interval comes from users (ddca_account.interval_in_seconds), which could be a large number such that the following integer arithmetic operations may overflow. For example, it would be safer to use checked_mul/checked_add instead of */+.

In addition, 10u128.pow at line 239 may overflow, However, its parameter comes from from_mint, it's worth considering replacing pow with pow_checked.

Resolution

The team acknowledged the findings. They made the changes to limit the range of ddca_account.interval_in_seconds and also replaced pow with pow_checked.

IMPACT – LOW

[DDCA-L-3] logic error when updating swap_avg_rate

```
/* ddca/programs/ddca/src/lib.rs */
ctx.accounts.ddca_account.swap_avg_rate = {
   if swap_rate >= ctx.accounts.ddca_account.swap_avg_rate {
       ctx.accounts.ddca_account.swap_avg_rate
       .checked_add(swap_rate).unwrap()
       .checked_sub(ctx.accounts.ddca_account.swap_avg_rate).unwrap()
       .checked_div(swap_count_plus_one).unwrap()
   }
   else {
       ctx.accounts.ddca_account.swap_avg_rate
       .checked_sub(ctx.accounts.ddca_account.swap_avg_rate).unwrap()
       .checked_sub(swap_rate).unwrap()
       .checked_div(swap_count_plus_one).unwrap()
   }
};
```

when `swap_rate >= ctx.accounts.ddca_account.swap_avg_rate`,

```
swap_avg_rate = ( (swap_avg_rate + swap_rate) - swap_avg_rate ) / swap_count_plus_one
                = swap_rate / swap_count_plus_one
```

Consider the following example

```
swap_rate = 120_u64
ctx.accounts.ddca_account.swap_avg_rate = 100_u64
swap_count_plus_one = 2_u64
```

`ctx.accounts.ddca_account.swap_avg_rate` will be updated to

```
   if swap_rate >= ctx.accounts.ddca_account.swap_avg_rate { // 120 > 100: true
       ctx.accounts.ddca_account.swap_avg_rate  // 100
       .checked_add(swap_rate).unwrap()         // 100 + 120 = 220
       .checked_sub(ctx.accounts.ddca_account.swap_avg_rate).unwrap() // 220-100=120
       .checked_div(swap_count_plus_one).unwrap()   // 120 / 2 = 60
```

when `swap_rate < ctx.accounts.ddca_account.swap_avg_rate`, the program will panic due to the integer underflow because

```
swap_avg_rate = ( (swap_avg_rate - swap_avg_rate) - swap_rate ) / swap_count_plus_one
```

Resolution

The team acknowledged the findings and fixed the issue.

IMPACT - INFO

[DDCA-I-1] partial account validation

Although DDCA and HLA together validate the accounts needed for the swap call and there is no issue for this particular case, each of them only partially validates some of the accounts. Since they are two standalone contracts, it would be a good idea for both to fully validate the accounts.

In particular, in DDCA, accounts `hla_operating_account` (`ddca/src/lib.rs:630`) and `hla_operating_from_token_account` (`ddca/src/lib.rs:636`) are not sufficiently validated such that it's possible to use a pair of faked accounts, although they will fail the account validation in HLA later.

```
/* ddca/programs/ddca/src/lib.rs */
157 | pub fn wake_and_swap<'info>(
158 |     ctx: Context<'_, '_, '_, 'info, WakeAndSwapInputAccounts<'info>>,
161 | ) -> ProgramResult {
200 |     // call hla to execute the first swap
201 |     let hla_cpi_program = ctx.accounts.hla_program.clone();
202 |     let hla_cpi_accounts = Swap {
203 |         hla_ops_account: ctx.accounts.hla_operating_account.clone(),
204 |         hla_ops_token_account: ctx.accounts.hla_operating_from_token_account...,
210 |         token_program_account: ctx.accounts.token_program...,
211 |     };
222 |     let hla_cpi_ctx = CpiContext::new(hla_cpi_program, hla_cpi_accounts)
227 |     hla::cpi::swap(hla_cpi_ctx, ...)?;

597 | #[derive(Accounts)]
598 | #[instruction(swap_min_out_amount: u64, swap_slippage: u64)]
599 | pub struct WakeAndSwapInputAccounts<'info> {
629 |     #[account(mut)]
630 |     pub hla_operating_account: AccountInfo<'info>,
631 |     #[account(
632 |         mut,
633 |         associated_token::mint = from_mint,
634 |         associated_token::authority = hla_operating_account,
635 |     )]
636 |     pub hla_operating_from_token_account: Box<Account<'info, TokenAccount>>,
641 |     pub token_program: Program<'info, Token>,
643 | }

/* hybrid-liquidity-ag/programs/hla/src/state.rs */
023 | #[derive(Accounts, Clone)]
031 | pub struct Swap<'info> {
040 |     #[account(mut, address = hla_ops_account::ID)]
```

```
    pub hla_ops_account: AccountInfo<'info>,
    #[account(mut)]
    pub hla_ops_token_account: AccountInfo<'info>,
    pub token_program_account: AccountInfo<'info>
}

/* hybrid-liquidity-ag/programs/hla/src/utils.rs */
pub fn get_transfer_context<'info>(
    swap_info: SwapInfo<'info>
) -> Result<CpiContext<'_, '_, '_, 'info, Transfer<'info>>> {
    let cpi_accounts = Transfer {
        to: swap_info.accounts.hla_ops_token_account.to_account_info(),
};
```

However, HLA requires hla_ops_account equals to hla_ops_account::ID (hla/src/state.rs:32). in order to successfully invoke swap in DDCA, hla_operating_account has to be hla_ops_account::ID (ddca/src/lib.rs:203) and hla_operating_from_token_account is fixed. On the other hand, hla_ops_token_account (hla/src/state.rs:35) is not validated, although it's bounded when DDCA calls HLA.

It may be a good idea to validate these accounts in both DDCA and HLA contracts.

Another example is token_program_account. It's not validated in HLA but is constrained in DDCA.

Resolution

The team acknowledged the findings. Validation for token accounts, mint accounts, HLA operating associated token account and token program account were added in the HLA program. Due to the tight computation budget, the team will perform more validations in the DDCA program in the future.

IMPACT – INFO

[DDCA-I-2] unused variable swap_slippage

The _slippage is not used in hla::swap so is the swap_slippage

```
/* ddca/programs/ddca/src/lib.rs */
pub fn wake_and_swap<'info>(
  swap_slippage: u64,
) -> ProgramResult {
    hla::cpi::swap(..., ..., ..., swap_slippage)?;

/* hybrid-liquidity-ag/programs/hla/src/lib.rs */
pub mod hla {
    pub fn swap<'info>(
        ctx: Context<'_, '_, '_, 'info, Swap<'info>>,
        from_amount: u64,
        min_out_amount: u64,
        _slippage: u64
    ) -> ProgramResult {
        // _slippage not used
    }
```

Resolution

The team acknowledged the finding and removed the variable from the HLA contract.

IMPACT – INFO

[DDCA-I-3] repeated statements

In function create in ddca/src/lib.rs, the same statement is repeated twice.

```
/* ddca/programs/ddca/src/lib.rs */
pub fn create(
  ctx: Context<CreateInputAccounts>,
) -> ProgramResult {
    ctx.accounts.ddca_account.owner_acc_addr = *ctx.accounts.owner_account.key;
    // ..., same scope
    ctx.accounts.ddca_account.owner_acc_addr = *ctx.accounts.owner_account.key;
```

Resolution

The team acknowledged the finding and removed the duplicated statement.

IMPACT – INFO

[DDCA-I-4] function create does not update laste_deposit_ts

In add_funds, after deposit, last_deposit_ts and last_deposit_slot are updated. However, in create, they are not. Is this an intended behavior? It seems these two variables are not used at other places.

```
pub fn add_funds(
    ctx: Context<AddFundsInputAccounts>,
    deposit_amount: u64,
) -> ProgramResult {
    ctx.accounts.ddca_account.last_deposit_ts = Clock::get()?.unix_timestamp as u64;
    ctx.accounts.ddca_account.last_deposit_slot =  Clock::get()?.slot;
}
```

Resolution

The team acknowledged the finding and updated last_deposit_ts in create too.

Contract HLA

IMPACT – INFO

[HLA-I-1] account validations

Please see [DDCA-I-1] for details.

Resolution

The team acknowledged the findings. Validation for token accounts, mint accounts, HLA operating associated token account and token program account were added in the HLA program. Due to the tight computation budget, the team will add more validations in the DDCA program in the future.

IMPACT – INFO

[HLA-I-2] dependencies versioning issues

Depending on the rust/cargo version, although `stable-swap-anchor 1.6.7` and `stable-swap-client 1.6.7` are specified, a newer version of `1.6.x` are picked up, which leads to errors such as `SwapUserContext does not have clock: clock_info.to_account_info().`

Resolution

The team acknowledged the finding and specified the exact versions used in `Cargo.toml`.

DISCLAIMER

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