Sec3™

Security Assessment Report

Jupiter Swap Aggregator v3.0.0

August 9th, 2022

Summary

The Sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the Jupiter Swap Aggregator v3.0.0 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 two contracts

- **Contract "Jupiter":**
    - Commit 64259b33ddbda7e5064cde97ef007472c1b8da60
    - Tag v3.0.0

The audit revealed 8 issues or questions. This report describes the findings and resolutions.

Table of Contents

Methodology and Scope of Work 3

Result Overview 4

Findings in Detail 5

[H-1] Manipulate swap in_amount using inconsistent token ledgers 5

[H-2] Arbitrary signed program invocation 8

[H-3] Unvalidated token program id 20

[M-1] User-controlled platform fee 21

[L-1] Fees are not rounded up 22

[I-1] Inconsistent refund account usage 23

[I-2] Orca Swap Program V1 id in raydium instruction project 24

[I-3] Best practice 25

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 Sec3 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    - Outdated dependencies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 JUPITER v3.0.0

Issue	Impact	Status
[H-1] Manipulate swap in_amount using inconsistent token ledgers	High	ACKED
[H-2] Arbitrary signed program invocation	High	Resolved
[H-3] Unvalidated token program id	High	Resolved
[M-1] User-controlled platform fee	Medium	Won't fix
[L-1] Fees are not rounded up	Low	Won't fix
[I-1] Inconsistent refund account usage	Informational	Resolved
[I-2] Orca Swap Program V1 id in raydium instruction project	Informational	Resolved
[I-3] Best practices	Informational	Won't fix

Findings in Detail

Impact Description

The impact of some issues listed in this report depends on how the swap instructions are built, which is not included in this on-chain program.

In particular, if all accounts and parameters are provided by end-users, it's equivalent to swapping directly using relevant external swap programs. The end-users can be blamed if incorrect inputs are used. However, if some intermediate accounts (e.g. swap program accounts or even intermediate token pool accounts) are provided by some off-chain components, end-users may not be responsible if something goes wrong.

Because we don't know how the instructions are constructed/signed and how trustworthy these off-chain components are, we assume these off-chain components cannot be trusted, which may be inconsistent with the threat model used by the Jupiter team.

IMPACT – HIGH

[H-1] Manipulate swap in_amount using inconsistent token ledgers

The token ledger accounts are used to compute the input amount for the next swap. However, the ledger accounts are not validated. Given anyone can create and prepare token ledger accounts, the computed swap amount can be inconsistent with the actual amount.

```
/* jupiter/programs/jupiter/src/lib.rs */
103 | macro_rules! process_swap {
104 |     ($ctx:expr, $protocol_swap:expr, $user_source_token_account:expr, ...) => {{
106 |         let (token_ledger, platform_fee_account) =
107 |             get_optional_accounts($in_amount, $platform_fee_bps, &mut remaining_accounts_iter);
109 |         let in_amount =
110 |             token_ledger::get_in_amount($in_amount, ..., token_ledger)?;
113 |         swap_wrapper(
114 |             || $protocol_swap(swap_ctx, in_amount, 0),
121 |         )
122 |     }};
123 | }
```

In particular, at line 109 in src/lib.rs, the token ledger account is used to determine the input amount for the swap that is initiated next.

```
/* jupiter/programs/jupiter/src/token_ledger.rs */
006 | pub fn get_in_amount(
007 |     in_amount: Option<u64>,
008 |     from_token_account: &AccountInfo,
009 |     token_ledger: Option<&AccountInfo>,
010 | ) -> Result<u64> {
011 |     match in_amount {
013 |         None => {
014 |             let token_ledger_account = token_ledger.ok_or(JupiterError::MissingTokenLedger)?;
015 |             let mut token_ledger: ... = Account::try_from(&token_ledger_account)?;
017 |             if token_ledger.token_account != *from_token_account.key {
018 |                 return Err(JupiterError::LedgerTokenAccountDoesNotMatch.into());
019 |             }
020 |             let current_amount = token::accessor::amount(from_token_account)?;
025 |             current_amount
026 |                 .checked_sub(token_ledger.amount)
028 |         }
029 |     }
030 | }

/* jupiter/programs/jupiter/src/lib.rs */
826 | pub fn set_token_ledger(ctx: Context<SetTokenLedger>) -> Result<()> {
827 |     ctx.accounts.token_ledger.token_account = ctx.accounts.token_account.key();
828 |     ctx.accounts.token_ledger.amount = ctx.accounts.token_account.amount;
829 |     Ok(())
830 | }
```

When in_amount is not provided, the input amount is calculated by current_amount - token_ledger.amount. This approach works when the token_ledger.amount is obtained right before the previous swap transfer. However, since there is no validation and anybody can create a token_ledger account that satisfies the check at line 17 in src/token_ledger.rs at any time using instruction set_token_ledger, it's possible to provide an outdated or inconsistent token ledge and manipulate the input.

Depending on the usage scenario and how the source token account is set up, especially when multiple hops are supported using token ledgers in the latest commit, inconsistent token_ledger accounts may lead to the following problems:

- For an intermediate step, when the `token_ledger.amount` provided is smaller than the actual amount, the input amount for the next hop will be larger. Depending on how the source token account is configured, the user may steal money from intermediate source accounts.
- For an intermediate step, when the `token_ledger.amount` provided is larger than the actual amount, the input amount for the next hop will be smaller than the actual amount. A portion of the token belonging to the user will not be swapped so the user gets less and loses money.

Resolution

The team acknowledged this finding and decided not to fix it. The team mentioned this is by design and are not sure how to mitigate this.

IMPACT – HIGH

[H-2] Arbitrary signed program invocation

The program ids of several CPIs are provided by users and not checked. As a result, users can control which program is invoked with signatures. For example, hackers (who set up the accounts in the instructions) may create a helper contract and provide the program id of this helper contract instead. In this way, users may be tricked (as it still interacts with the Jupiter contract) and sign the transaction. However, instead of swapping tokens, the hacker's contract may directly transfer and steal the token from the user accounts.

More information about this issue and fixes could be found @ https://github.com/coral-xyz/sealevel-attacks/tree/master/programs/5-arbitrary-cpi

1. TokenSwap

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct TokenSwap<'info> {
    pub token_swap_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn token_swap<'info>(
    ctx: Context<'_, '_, '_, 'info, TokenSwap<'info>>,
) -> Result<()> {
    process_swap!(
        token_swap_amm_swap,
    )
}

/* jupiter/programs/jupiter/src/lib.rs */
use amm::{
    token_swap::swap as token_swap_amm_swap,
};

/* jupiter/programs/jupiter/src/amm/token_swap.rs */
fn from(accounts: &mut TokenSwap<'info>) -> CpiContext<'a, 'b, 'c, 'info, TokenSwap<'info>> {
    let cpi_accounts = TokenSwap {
        token_swap_program: accounts.token_swap_program.clone(),
    };
    CpiContext::new(..., cpi_accounts)
}
```

```
/* jupiter/programs/jupiter/src/amm/token_swap.rs */
pub fn swap<'a, 'b, 'c, 'info>(
    ctx: CpiContext<'a, 'b, 'c, 'info, TokenSwap<'info>>,
) -> Result<()> {
    let ix = spl_token_swap::instruction::swap(
        ctx.accounts.token_swap_program.key,
    )?;
    solana_program::program::invoke(&ix, ...)
}


/* spl-token-swap-2.1.0/src/instruction.rs */
pub fn swap(
    program_id: &Pubkey,
) -> Result<Instruction, ProgramError> {
    Ok(Instruction {
        program_id: *program_id, // program_id unvalidated
    })
}
```

2. StepTokenSwap

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct StepTokenSwap<'info> {
    pub token_swap_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn step_token_swap<'info>(
    ctx: Context<'_, '_, '_, 'info, StepTokenSwap<'info>>,
) -> Result<()> {
    process_swap!(
        step_swap,
    )
}

/* jupiter/programs/jupiter/src/lib.rs */
use amm::{
    step::swap as step_swap,
};

/* jupiter/programs/jupiter/src/amm/step.rs */
fn from(
    accounts: &mut StepTokenSwap<'info>,
) -> CpiContext<'a, 'b, 'c, 'info, StepTokenSwap<'info>> {
    let cpi_accounts = StepTokenSwap {
        token_swap_program: accounts.token_swap_program.clone(),
    };
    CpiContext::new(..., cpi_accounts)
}
```

```
/* jupiter/programs/jupiter/src/amm/step.rs */
pub fn swap<'a, 'b, 'c, 'info>(
    ctx: CpiContext<'a, 'b, 'c, 'info, StepTokenSwap<'info>>,
) -> Result<()> {
    let ix = solana_program::instruction::Instruction {
        program_id: *ctx.accounts.token_swap_program.key,  // program_id unvalidated
    };
    solana_program::program::invoke(&ix, ...)
}
```

3. CropperSwap

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct CropperSwap<'info> {
    pub token_swap_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn cropper_token_swap<'info>(
    ctx: Context<'_, '_, '_, 'info, CropperSwap<'info>>,
) -> Result<()> {
    process_swap!(
        cropper_swap,
    )
}

/* jupiter/programs/jupiter/src/lib.rs */
use amm::{
    cropper::swap as cropper_swap,
};

/* jupiter/programs/jupiter/src/amm/cropper.rs */
fn from(
    accounts: &mut CropperSwap<'info>,
) -> CpiContext<'a, 'b, 'c, 'info, CropperSwap<'info>> {
    let cpi_accounts = CropperSwap {
        token_swap_program: accounts.token_swap_program.clone(),
    };
    CpiContext::new(..., cpi_accounts)
}

/* jupiter/programs/jupiter/src/amm/cropper.rs */
pub fn swap<'a, 'b, 'c, 'info>(
    ctx: CpiContext<'a, 'b, 'c, 'info, CropperSwap<'info>>,
) -> Result<()> {
    let ix = solana_program::instruction::Instruction {
        program_id: *ctx.accounts.token_swap_program.key,  // program_id unvalidated
    };
    solana_program::program::invoke(&ix, ...)
}
```

4. RaydiumSwap & RaydiumSwapV2

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct RaydiumSwap<'info> {
    pub swap_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn raydium_swap<'info>(
    ctx: Context<'_, '_, '_, 'info, RaydiumSwap<'info>>,
) -> Result<()> {
    process_swap!(
        raydium_amm_swap,
    )
}

/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct RaydiumSwapV2<'info> {
    pub swap_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn raydium_swap_v2<'info>(
    ctx: Context<'_, '_, '_, 'info, RaydiumSwapV2<'info>>,
) -> Result<()> {
    process_swap!(
        raydium_amm_swap,
    )
}

/* jupiter/programs/jupiter/src/lib.rs */
use amm::{
    raydium::swap as raydium_amm_swap,
};

/* jupiter/programs/jupiter/src/amm/raydium.rs */
fn from(
    accounts: &mut RaydiumSwap<'info>,
) -> CpiContext<'a, 'b, 'c, 'info, RaydiumSwapV2<'info>> {
    let cpi_accounts = RaydiumSwapV2 {
        swap_program: accounts.swap_program.clone(),
    };
    CpiContext::new(..., cpi_accounts)
}

/* jupiter/programs/jupiter/src/amm/raydium.rs */
fn from(
    accounts: &mut RaydiumSwapV2<'info>,
) -> CpiContext<'a, 'b, 'c, 'info, RaydiumSwapV2<'info>> {
    let cpi_accounts = RaydiumSwapV2 {
        swap_program: accounts.swap_program.clone(),
    };
    CpiContext::new(..., cpi_accounts)
}

/* jupiter/programs/jupiter/src/amm/raydium.rs */
pub fn swap<'a, 'b, 'c, 'info>(
    ctx: CpiContext<'a, 'b, 'c, 'info, RaydiumSwapV2<'info>>,
    in_amount: u64,
    minimum_out_amount: u64,
) -> Result<()> {
    let ix = raydium::instruction::swap(
        ctx.accounts.swap_program.key,
    )?;
    solana_program::program::invoke(&ix, ...)
}

/* jupiter/programs/raydium/src/instruction.rs */
pub fn swap(
    program_id: &Pubkey,
) -> Result<Instruction, ProgramError> {
    Ok(Instruction {
        program_id: *program_id,  // program_id unvalidated
    })
}
```

5. AldrinSwap

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct AldrinSwap<'info> {
    pub swap_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn aldrin_swap<'info>(
    ctx: Context<'_, '_, '_, 'info, AldrinSwap<'info>>,
) -> Result<()> {
    process_swap!(
        |swap_ctx, in_amount, minimum_out_amount| aldrin_amm_swap
    )
}

/* jupiter/programs/jupiter/src/lib.rs */
use amm::{
    aldrin::swap as aldrin_amm_swap,
};
```

```
/* jupiter/programs/jupiter/src/amm/aldrin.rs */
fn from(accounts: &mut AldrinSwap<'info>) -> CpiContext<'a, 'b, 'c, 'info, AldrinSwap<'info>> {
    let cpi_accounts = AldrinSwap {
        swap_program: accounts.swap_program.clone(),
    };
    CpiContext::new(..., cpi_accounts)
}

/* jupiter/programs/jupiter/src/amm/aldrin.rs */
pub fn swap<'a, 'b, 'c, 'info>(
    ctx: CpiContext<'a, 'b, 'c, 'info, AldrinSwap<'info>>,
) -> Result<()> {
    let ix = solana_program::instruction::Instruction {
        program_id: *ctx.accounts.swap_program.key,  // program_id unvalidated
    };
    solana_program::program::invoke(&ix, ...)
}
```

6. AldrinV2Swap

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct AldrinV2Swap<'info> {
    pub swap_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn aldrin_v2_swap<'info>(
    ctx: Context<'_, '_, '_, 'info, AldrinV2Swap<'info>>,
) -> Result<()> {
    process_swap!(
        ctx,
        |swap_ctx, in_amount, minimum_out_amount| aldrin_v2_amm_swap
    )
}

/* jupiter/programs/jupiter/src/lib.rs */
use amm::{
    aldrin_v2::swap as aldrin_v2_amm_swap,
};

/* jupiter/programs/jupiter/src/amm/aldrin_v2.rs */
fn from(
    accounts: &mut AldrinV2Swap<'info>,
) -> CpiContext<'a, 'b, 'c, 'info, AldrinV2Swap<'info>> {
    let cpi_accounts = AldrinV2Swap {
        swap_program: accounts.swap_program.clone(),
    };
    CpiContext::new(accounts.swap_program.to_account_info(), cpi_accounts)
}
```

```
/* jupiter/programs/jupiter/src/amm/aldrin_v2.rs */
pub fn swap<'a, 'b, 'c, 'info>(
    ctx: CpiContext<'a, 'b, 'c, 'info, AldrinV2Swap<'info>>,
) -> Result<()> {
    let ix = solana_program::instruction::Instruction {
        program_id: *ctx.accounts.swap_program.key,  // program_id unvalidated
    };
    solana_program::program::invoke(&ix, ...)
}
```

7. CremaSwap

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct CremaSwap<'info> {
    pub swap_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn crema_token_swap<'info>(
    ctx: Context<'_, '_, '_, 'info, CremaSwap<'info>>,
) -> Result<()> {
    process_swap!(
        crema_swap,
    )
}

/* jupiter/programs/jupiter/src/lib.rs */
use amm::{
    crema::swap as crema_swap,
};

/* jupiter/programs/jupiter/src/amm/crema.rs */
fn from(accounts: &mut CremaSwap<'info>) -> CpiContext<'a, 'b, 'c, 'info, CremaSwap<'info>> {
    let cpi_accounts = CremaSwap {
        swap_program: accounts.swap_program.clone(),
    };
    CpiContext::new(..., cpi_accounts)
}

/* jupiter/programs/jupiter/src/amm/crema.rs */
pub fn swap<'a, 'b, 'c, 'info>(
    ctx: CpiContext<'a, 'b, 'c, 'info, CremaSwap<'info>>,
) -> Result<()> {
    let ix = solana_program::instruction::Instruction {
        program_id: *ctx.accounts.swap_program.key,  // program_id unvalidated
    };
    solana_program::program::invoke(&ix, ...)
}
```

8. LifinitySwap

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct LifinitySwap<'info> {
    pub swap_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn lifinity_token_swap<'info>(
    ctx: Context<'_, '_, '_, 'info, LifinitySwap<'info>>,
) -> Result<()> {
    process_swap!(
        lifinity_swap,
    )
}

/* jupiter/programs/jupiter/src/lib.rs */
use amm::{
    lifinity::swap as lifinity_swap,
};

/* jupiter/programs/jupiter/src/amm/lifinity.rs */
fn from(
    accounts: &mut LifinitySwap<'info>,
) -> CpiContext<'a, 'b, 'c, 'info, LifinitySwap<'info>> {
    let cpi_accounts = LifinitySwap {
        swap_program: accounts.swap_program.clone(),
    };
    CpiContext::new(..., cpi_accounts)
}

/* jupiter/programs/jupiter/src/amm/lifinity.rs */
pub fn swap<'a, 'b, 'c, 'info>(
    ctx: CpiContext<'a, 'b, 'c, 'info, LifinitySwap<'info>>,
) -> Result<()> {
    let ix = solana_program::instruction::Instruction {
        program_id: *ctx.accounts.swap_program.key,  // program_id unvalidated
    };
    solana_program::program::invoke(&ix, ...)
}
```

9. CykuraSwap

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct CykuraSwap<'info> {
    pub core_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
```

```
pub fn cykura_swap<'info>(
    ctx: Context<'_, '_, '_, 'info, CykuraSwap<'info>>,
) -> Result<()> {
    let cpi_accounts = ExactInputSingle {
        core_program: ctx.accounts.core_program.to_account_info(),
    };
    let swap_ctx = CpiContext::new(ctx.accounts.swap_program.to_account_info(), cpi_accounts)
    swap_wrapper(
        || exact_input_single(swap_ctx, std::i64::MAX, in_amount, 0, 0),
    )
}

/* cyclos-core-0.1.0/src/lib.rs */
pub fn exact_input_single<'a, 'b, 'c, 'info>(
    ctx: Context<'a, 'b, 'c, 'info, ExactInputSingle<'info>>,
) -> Result<()> {
    let amount_out = exact_input_internal(
        &mut SwapContext {
            callback_handler: UncheckedAccount::try_from(
                ctx.accounts.core_program.to_account_info(),
            ),
        },
    )?;
}

/* cyclos-core-0.1.0/src/lib.rs */
pub fn exact_input_internal<'info>(
    accounts: &mut SwapContext<'info>,
) -> Result<u64> {
    swap(
        Context::new(&ID, accounts, remaining_accounts, BTreeMap::default()),
    )?;
}

/* cyclos-core-0.1.0/src/lib.rs */
pub fn swap(
    ctx: Context<SwapContext>,
) -> Result<()> {
    if zero_for_one {
        // transfer tokens to pool in callback
        let ix = Instruction::new_with_bytes(
            ctx.accounts.callback_handler.key(),  // program_id unvalidated
        );
        solana_program::program::invoke(&ix, &ctx.accounts.to_account_infos())?;
    } else {
        // transfer tokens to pool in callback
        let ix = Instruction::new_with_bytes(
            ctx.accounts.callback_handler.key(),  // program_id unvalidated
        );
        solana_program::program::invoke(&ix, &ctx.accounts.to_account_infos())?;
    }
}
```

10. WhirlpoolSwap

- `whirlpool_swap()`
- `whirlpool_swap_exact_output()`

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct WhirlpoolSwap<'info> {
    pub swap_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn whirlpool_swap<'info>(
    ctx: Context<'_, '_, '_, 'info, WhirlpoolSwap<'info>>,
) -> Result<()> {
    process_swap!(
        ctx,
        |swap_ctx, in_amount, minimum_out_amount| whirlpool::swap
    )
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn whirlpool_swap_exact_output<'info>(
    ctx: Context<'_, '_, '_, 'info, WhirlpoolSwap<'info>>,
) -> Result<()> {
    process_swap_exact_output!(
        ctx,
        |swap_ctx, out_amount, maximum_in_amount| whirlpool::swap
    )
}

/* jupiter/programs/jupiter/src/amm/whirlpool.rs */
fn from(
    accounts: &mut WhirlpoolSwap<'info>,
) -> CpiContext<'a, 'b, 'c, 'info, WhirlpoolSwap<'info>> {
    let cpi_accounts = WhirlpoolSwap {
        swap_program: accounts.swap_program.clone(),
    };
    CpiContext::new(accounts.swap_program.to_account_info(), cpi_accounts)
}

/* jupiter/programs/jupiter/src/amm/whirlpool.rs */
pub fn swap<'a, 'b, 'c, 'info>(
    ctx: CpiContext<'a, 'b, 'c, 'info, WhirlpoolSwap<'info>>,
    a_to_b: bool,
) -> Result<()> {
    let ix = solana_program::instruction::Instruction {
        program_id: *ctx.accounts.swap_program.key,  // program_id unvalidated
    };
    solana_program::program::invoke(&ix, &ctx.accounts.to_account_infos())
}
```

11. MarinadeFinanceDeposit

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct MarinadeFinanceDeposit<'info> {
    pub marinade_finance_program: UncheckedAccount<'info>,
}

/* jupiter/programs/jupiter/src/lib.rs */
pub fn marinade_finance_deposit<'info>(
    ctx: Context<'_, '_, '_, 'info, MarinadeFinanceDeposit<'info>>,
    in_amount: Option<u64>,
    minimum_out_amount: u64,
    platform_fee_bps: u8,
) -> Result<()> {
    process_swap!(
        |swap_ctx, in_amount, _minimum_out_amount| {
            marinade_finance::deposit
        }
    )
}

/* jupiter/programs/jupiter/src/amm/marinade_finance.rs */
fn from(
    accounts: &mut MarinadeFinanceDeposit<'info>,
) -> CpiContext<'a, 'b, 'c, 'info, MarinadeFinanceDeposit<'info>> {
    let cpi_accounts = MarinadeFinanceDeposit {
        marinade_finance_program: accounts.marinade_finance_program.clone(),
    };
    CpiContext::new(
        cpi_accounts,
    )
}

/* jupiter/programs/jupiter/src/amm/marinade_finance.rs */
pub fn deposit<'a, 'b, 'c, 'info>(
    ctx: CpiContext<'a, 'b, 'c, 'info, MarinadeFinanceDeposit<'info>>,
) -> Result<()> {
    let ix = solana_program::instruction::Instruction {
        program_id: *ctx.accounts.marinade_finance_program.key,  // program_id unvalidated
    };
    solana_program::program::invoke_signed(&ix, ..., ...)
}
```

12. MarinadeFinanceLiquidUnstake

```
/* jupiter/programs/jupiter/src/account_structs.rs */
#[derive(Accounts)]
pub struct MarinadeFinanceLiquidUnstake<'info> {
    pub marinade_finance_program: UncheckedAccount<'info>,
}
```

```
/* jupiter/programs/jupiter/src/lib.rs */
pub fn marinade_finance_liquid_unstake<'info>(
    ctx: Context<'_, '_, '_, 'info, MarinadeFinanceLiquidUnstake<'info>>,
) -> Result<()> {
    process_swap!(
        |swap_ctx, in_amount, _minimum_out_amount| {
            marinade_finance::liquid_unstake(swap_ctx, in_amount)?;
        },
    )
}

/* jupiter/programs/jupiter/src/amm/marinade_finance.rs */
fn from(
    accounts: &mut MarinadeFinanceLiquidUnstake<'info>,
) -> CpiContext<'a, 'b, 'c, 'info, MarinadeFinanceLiquidUnstake<'info>> {
    let cpi_accounts = MarinadeFinanceLiquidUnstake {
        marinade_finance_program: accounts.marinade_finance_program.clone(),
    };
    CpiContext::new(
        cpi_accounts,
    )
}

/* jupiter/programs/jupiter/src/amm/marinade_finance.rs */
pub fn liquid_unstake<'a, 'b, 'c, 'info>(
    ctx: CpiContext<'a, 'b, 'c, 'info, MarinadeFinanceLiquidUnstake<'info>>,
) -> Result<()> {
    let ix = solana_program::instruction::Instruction {
        program_id: *ctx.accounts.marinade_finance_program.key,  // program_id unvalidated
    };
```

Resolution

The program id validations have been added, expect the one in `TokenSwap` to support more spl-token-swap forks. This issue has been resolved.

IMPACT – HIGH

[H-3] Unvalidated token program id

The validation of the token program id accounts is not in this contract and depends on the external functions/contracts.

When creating instructions using helper functions such as spl_token::instruction::transfer, the token program id is validated inside these helper functions. However, the source code of some external instructions is not available (e.g. mercurial_exchange and cropper_swap). It's not clear if the token program id is validated in the external instructions.

In general, it's safer to validate the token program id via "pub token_program: Program<'info, Token>". Similarly, the system program id can be validated using "pub system_program: Program<'info, System>".

Resolution

The token program id validations are added. This issue is resolved.

IMPACT – MEDIUM

[M-1] User-controlled platform fee

The user-provided instruction argument `platform_fee_bps` can be determined and hence skipped by users in all swap instructions. For example,

```
/* jupiter/programs/jupiter/src/lib.rs */
pub fn mercurial_exchange<'info>(
    ctx: Context<'_, '_, '_, 'info, MercurialExchange<'info>>,
    in_amount: Option<u64>,
    minimum_out_amount: u64,
    platform_fee_bps: u8,
) -> Result<()>

/* jupiter/programs/jupiter/src/lib.rs */
pub fn saber_exchange<'info>(
    ctx: Context<'_, '_, '_, 'info, SaberSwapLegacy<'info>>,
    in_amount: Option<u64>,
    minimum_out_amount: u64,
    platform_fee_bps: u8,
) -> Result<()>
```

Although the impact of this issue may be limited depending on how the off-chain components use the APIs, ideally, the platform fee would be configurable constants stored on-chain (the drawback of course is that this is less flexible and efficient).

Resolution

This is an intended behavior by design and won't be fixed.

IMPACT – LOW

[L-1] Fees are not rounded up

The fees may be less than the actual amount due to truncated division.

```rust
/* jupiter/programs/jupiter/src/fees.rs */
pub fn calculate_fees(trading_tokens: u64, fee_bps: u8) -> Option<u64> {
    let fee = u128::from(trading_tokens)
        .checked_mul(u128::from(fee_bps))?
        .checked_div(PLATFORM_FEE_DENOMINATOR)?;

    Some(fee.try_into().ok()?)
}

pub fn apply_platform_fee<'info>(
) -> Result<()> {
    let platform_fee_account =
        platform_fee_account.ok_or(JupiterError::MissingPlatformFeeAccount)?;

    let fee = calculate_fees(trading_tokens, fee_bps).ok_or(JupiterError::InvalidCalculation)?;
    token::transfer(
        CpiContext::new(
            token_program.clone(),
            Transfer {
                from: user_token_account.clone(),
                to: platform_fee_account.clone(),
                authority: user_transfer_authority.clone(),
            },
        ),
        fee,
    )
}
```

Resolution

This is an intended behavior by design and won't be fixed.

IMPACT – INFO

[I-1] Inconsistent refund account usage

The comment on line 29 and the account provided on line 31 seem inconsistent with the underlying implementation.

```
/* jupiter/programs/jupiter/src/amm/step.rs */
005 | pub fn swap<'a, 'b, 'c, 'info>(
009 | ) -> Result<()> {
010 |     let mut data =
011 |         spl_token_swap::instruction::SwapInstruction::Swap(spl_token_swap::instruction::Swap {
014 |         })
019 |     let accounts = vec![
020 |         AccountMeta::new_readonly(*ctx.accounts.swap.key, false),          // Acct 0
029 |         // Refund account, does not matter as untouched, repeat something else
030 |         AccountMeta::new_readonly(*ctx.accounts.token_program.key, false), // Acct 9
031 |         AccountMeta::new_readonly(*ctx.accounts.token_program.key, false), // Acct 10, it's used in swap
032 |     ];

/* spl-token-swap-2.1.0/src/instruction.rs */
113 | ///   Swap the tokens in the pool.
114 | ///
115 | ///   0. `[]` Token-swap
124 | ///   9. '[]` Token program id
125 | ///   10 `[optional, writable]` Host fee account to receive additional trading fees
126 | Swap(Swap),

/* spl-token-swap-2.1.0/src/processor.rs */
326 | pub fn process_swap(
331 | ) -> ProgramResult {
332 |     let account_info_iter = &mut accounts.iter();
342 |     let token_program_info = next_account_info(account_info_iter)?;
441 |     if pool_token_amount > 0 {
443 |         if let Ok(host_fee_account_info) = next_account_info(account_info_iter) { // refund acct used here
444 |             let host_fee_account = Self::unpack_token_account(
445 |                 host_fee_account_info,
447 |             )?;
455 |             if host_fee > 0 {
459 |                 Self::token_mint_to(
463 |                     host_fee_account_info.clone(),
```

Resolution

This is an intended behavior by design. A comment is added to clarify the usage.

IMPACT – INFO

[I-2] Orca Swap Program V1 id in raydium instruction project

```
/* jupiter/programs/raydium/src/lib.rs */
solana_program::declare_id!("DjVE6JNiYqPL2QXyCUUh8rNjHrbz9hXHNYt99MQ59qw1");
```

Resolution

The program id has been fixed.

IMPACT – INFO

[I-3] Best practice

1. Unchecked Token Accounts and Authority Accounts

Most authority and token accounts are not marked as <Signer> or <Token> types with the assumption that they're validated downstream. It's better to check and fail early before making the CPI.

2. Unchecked Math Operation

There is an unchecked addition in the Raydium unpack function

```
/* jupiter/programs/raydium/src/instruction.rs */
1082 | pub fn unpack<T>(input: &[u8]) -> Result<&T, ProgramError> {
1083 |     if input.len() < size_of::<u8>() + size_of::<T>() {
1084 |         return Err(ProgramError::InvalidAccountData);
1085 |     }
```

Not a significant issue given size_of won't return a large enough value to overflow but for consistent code practices, checked math operations should be used.

3. Inconsistencies use of AccountInfo vs UncheckedAccount

Resolution

The team decided not to fix them.

DISCLAIMER

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

Sec3™