SHERLOCK

# SHERLOCK SECURITY REVIEW FOR

|  |  |
|---|---|
| Contest type: | Public |
| Prepared for: | WooFi |
| Prepared by: | Sherlock |
| Lead Security Expert: | g |
| Dates Audited: | September 16 - September 24, 2024 |
| Prepared on: | October 23, 2024 |

## Introduction

This contest is for the Solana deployment of WOOFi's sPMM, which has been running on 10+ EVM chains.

### Scope

Repository: woonetwork/WOOFi_Solana

Branch: main

Audited Commit: c7835fbafdb3fe154b2365fea1969058caa9ee09

Final Commit: 6b27c4680b25cc6a6fda6462d22dd670dbfe62a6

---

For the detailed scope, see the contest details.

### Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

### Issues found

| Medium | High |
|---|---|
| 3 | 2 |

### Issues not fixed or acknowledged

| Medium | High |
|---|---|
| 0 | 0 |

### Security experts who found valid issues

| | | |
|---|---|---|
| g | D1r3Wolf | LZ_security |
| S3v3ru5 | Albort | dod4ufn |
| shaflow01 | 0xeix | Q7 |
| zigtur | calc1f4r | |

## Issue H-1: `rebate_info` and `rebate_manager` are unable to sign the CPI call due to an incorrect implementation of the `seeds` function

Source: https://github.com/sherlock-audit/2024-08-woofi-solana-deployment-judging/issues/29

### Found by

g, shaflow01

### Summary

`rebate_info` and `rebate_manager` will not be able to sign the CPI message because their `seeds` function has been implemented incorrectly.

### Root Cause

The implementation of the seeds function is incorrect because the correct seed needs to include the full seed phrase and the bump, but the seeds function does not include the bump.

https://github.com/sherlock-audit/2024-08-woofi-solana-deployment/blob/1c4c9c622e8c44ae2f8cd4219c7c2a0181f25ca0/WOOFi_Solana/programs/rebate_manager/src/state/rebate_manager.rs#L54

```
pub fn seeds(&self) -> [&[u8]; 2] {
    [
        REBATEMANAGER_SEED.as_bytes(),
        self.quote_token_mint.as_ref(),
    ]
}
```

https://github.com/sherlock-audit/2024-08-woofi-solana-deployment/blob/1c4c9c622e8c44ae2f8cd4219c7c2a0181f25ca0/WOOFi_Solana/programs/rebate_manager/src/state/rebate_info.rs#L51

```
pub fn seeds(&self) -> [&[u8]; 3] {
    [
        REBATEINFO_SEED.as_bytes(),
        self.rebate_manager.as_ref(),
        self.rebate_authority.as_ref(),
    ]
}
```

## Internal pre-conditions

None

## External pre-conditions

None

## Attack Path

None

## Impact

It will prevent the `claim_rebate_fee` and `withdraw` operations from executing, resulting in tokens being permanently locked in the contract.

## PoC

*No response*

## Mitigation

Here is an example of fixing rebate_manager:

```
#[account]
#[derive(InitSpace)]
pub struct RebateManager {
    pub authority: Pubkey, // 32

    #[max_len(ADMIN_AUTH_MAX_LEN)]
    pub admin_authority: Vec<Pubkey>,

    pub quote_token_mint: Pubkey, // 32

    pub token_vault: Pubkey, // 32

+   pub rebate_manager_bump: [u8; 1],
}
```

```
    pub fn seeds(&self) -> [&[u8]; 2] {
        [
            REBATEMANAGER_SEED.as_bytes(),
            self.quote_token_mint.as_ref(),
+           self.rebate_manager_bump.as_ref(),
        ]
    }
```

```
    pub fn initialize(
        &mut self,
        authority: Pubkey,
        quote_token_mint: Pubkey,
        token_vault: Pubkey,
+       bump: u8.
    ) -> Result<()> {
        self.authority = authority;
        self.quote_token_mint = quote_token_mint;
        self.token_vault = token_vault;
+       self.rebate_manager_bump = [bump];
        Ok(())
    }
```

```
pub fn handler(ctx: Context<CreateRebateManager>) -> Result<()> {
    let authority = ctx.accounts.authority.key();
    let quote_token_mint = ctx.accounts.quote_token_mint.key();
    let token_vault = ctx.accounts.token_vault.key();
+   let bump = ctx.bumps.rebate_manager;
    let rebate_manager = &mut ctx.accounts.rebate_manager;

-   rebate_manager.initialize(authority, quote_token_mint, token_vault);
+   rebate_manager.initialize(authority, quote_token_mint, token_vault, bump);
}
```

The fix for `rebate_info` is the same as described above.

## Discussion

**toprince**

Valid. same with https://github.com/sherlock-audit/2024-08-woofi-solana-deployment-judging/issues/18

**sherlock-admin2**

The protocol team fixed this issue in the following PRs/commits:
https://github.com/woonetwork/WOOFi_Solana/pull/32

**gjaldon**

The `rebate_manager`'s bump is now stored in the account and included in its `seeds()`. The issue is fixed and transfers from the `rebate_manager` will succeed.

## Issue H-2: Quote pools are expected to have same base token and quote token but this is not enforced in swaps

Source: https://github.com/sherlock-audit/2024-08-woofi-solana-deployment-judging/issues/64

### Found by

S3v3ru5, g

### Summary

The missing constraint that enforces quote pools should have the same base and quote token will cause swap fees to be deducted from non-quote pools.

By design, quote pools are pools with the same base and quote token. The development team has confirmed this. All swap fees should come from quote pools.

```
// record fee into account
woopool_quote.sub_reserve(swap_fee).unwrap();
woopool_quote.add_unclaimed_fee(swap_fee).unwrap();
```

### Root Cause

In swap.rs:79-84, there is no constraint that enforces that the pool used as the quote pool has the same base token and quote token.

```
#[account(mut,
    has_one = wooconfig,
    constraint = woopool_quote.token_mint == woopool_from.quote_token_mint,
    constraint = woopool_quote.authority == woopool_from.authority,
)]
woopool_quote: Box<Account<'info, WooPool>>,
```

This means that non-quote pools can be used as quote pools during swaps and swap fees will be deducted from these.

```
// record fee into account
woopool_quote.sub_reserve(swap_fee).unwrap();
woopool_quote.add_unclaimed_fee(swap_fee).unwrap();
```

### Internal pre-conditions

None

## External pre-conditions

None

## Attack Path

1. Anyone can execute swaps by invoking the swap instruction and passing a non-quote pool as a `woopool_quote`. The constraints will allow it as long as the `woopool_quote`'s base token is the same as the `woopool_from`'s quote token and the pools have the same owners.

## Impact

Swap fees can be deducted from non-quote pools instead of quote pools only.

## PoC

*No response*

## Mitigation

Consider adding a constraint that enforces that the quote pool is a pool with the same base and quote token.

## Discussion

**toprince**

Need investigation here. Same with https://github.com/sherlock-audit/2024-08-woofi-solana-deployment-judging/issues/40

**toprince**

should be Not valid.

1. all pools' quote_token_mint is USDC after create pool
2. constraint = woopool_quote.token_mint == woopool_from.quote_token_mint
3. above constraint should make sure the quote pool is USDC

**sherlock-admin2**

The protocol team fixed this issue in the following PRs/commits: https://github.com/woonetwork/WOOFi_Solana/pull/29

**gjaldon**

The added constraints to `swap()` fixes the issue.

## Issue M-1: An admin authority initializing RebateInfo will make claim_rebate_fee unusable

Source: https://github.com/sherlock-audit/2024-08-woofi-solana-deployment-judging/issues/13

### Found by

Albort, D1r3Wolf, g, zigtur

### Summary

A `ClaimRebateFee` constraint enforces that `rebate_info.authority == rebate_manager.authority`. This will always be false when an admin authority initialized the `rebate_info`, leading the `rebate_info.rebate_authority` to not be able to claim their rebate fee.

### Root Cause

In claim_rebate_fee.rs:26, there is an incorrect constraint.

### Internal pre-conditions

1. An admin authority needs to initialize the `rebate_info` through the `create_rebate_info` instruction. It is made possible through the constraint at create_rebate_info.rs#L17.

### External pre-conditions

None.

### Attack Path

*No response*

### Impact

- The rebate authority suffers from 100% rebate fee loss as it is not able to claim (through the `claim_rebate_fee` instruction).

### PoC

*No response*

## Mitigation

This constraint should be deleted. Fixing it to check if the `rebate_info.authority` is an admin authority will lead to the same issue being triggered when admin authorities are updated.

## Discussion

**toprince**

Need further investigation.

**sherlock-admin2**

The protocol team fixed this issue in the following PRs/commits:
https://github.com/woonetwork/WOOFi_Solana/pull/28

**gjaldon**

Issue is fixed. This change fixes the issue by also allowing admin authorities to claim rebate fees.

## Issue M-2: Missing permission control in create_oracle and create_pool.

Source: https://github.com/sherlock-audit/2024-08-woofi-solana-deployment-judging/issues/54

### Found by

0xeix, LZ_security, Q7, S3v3ru5, calc1f4r, dod4ufn, g, shaflow01, zigtur

### Summary

Missing permission control in create_oracle and create_pool.

### Vulnerability Detail

```
#[derive(Accounts)]
pub struct CreateWooracle<'info> {
    pub wooconfig: Box<Account<'info, WooConfig>>,
    pub token_mint: Account<'info, Mint>,

    #[account(
        init,
        payer = admin,
        space = 8 + Wooracle::INIT_SPACE,
        seeds = [
            WOORACLE_SEED.as_bytes(),
            wooconfig.key().as_ref(),
            token_mint.key().as_ref(),
            feed_account.key().as_ref(),
            price_update.key().as_ref()
            ],
        bump,
    )]
    wooracle: Account<'info, Wooracle>,
    #[account(mut)]
@>>    admin: Signer<'info>,
    system_program: Program<'info, System>,
    /// CHECK: This is the Pyth feed account
    feed_account: AccountInfo<'info>,
    // Add this account to any instruction Context that needs price data.
    // Warning:
    // users must ensure that the account passed to their instruction is owned
↪   by the Pyth pull oracle program.
```

```
    // Using Anchor with the Account<'info, PriceUpdateV2> type will
↪   automatically perform this check.
    // However, if you are not using Anchor, it is your responsibility to
↪   perform this check.
    price_update: Account<'info, PriceUpdateV2>,

    quote_token_mint: Account<'info, Mint>,
    /// CHECK: This is the Quote token's pyth feed account
    quote_feed_account: AccountInfo<'info>,
    // Add this account to any instruction Context that needs price data.
    // Warning:
    // users must ensure that the account passed to their instruction is owned
↪   by the Pyth pull oracle program.
    // Using Anchor with the Account<'info, PriceUpdateV2> type will
↪   automatically perform this check.
    // However, if you are not using Anchor, it is your responsibility to
↪   perform this check.
    quote_price_update: Account<'info, PriceUpdateV2>,
}

pub fn handler(ctx: Context<CreateWooracle>, maximum_age: u64) -> Result<()> {
    ctx.accounts.wooracle.wooconfig = ctx.accounts.wooconfig.key();
@>>    ctx.accounts.wooracle.authority = ctx.accounts.admin.key();
    ctx.accounts.wooracle.token_mint = ctx.accounts.token_mint.key();
    ctx.accounts.wooracle.feed_account = ctx.accounts.feed_account.key();
    ctx.accounts.wooracle.price_update = ctx.accounts.price_update.key();

   //----skip
}
```

From the code, it is evident that create_wooracle lacks permission control, allowing anyone to create an oracle.

```
#[derive(Accounts)]
pub struct CreatePool<'info> {
    pub wooconfig: Box<Account<'info, WooConfig>>,
    pub token_mint: Account<'info, Mint>,
    pub quote_token_mint: Account<'info, Mint>,

    #[account(mut)]
@>>    pub authority: Signer<'info>,

    #[account(
        init,
        payer = authority,
        space = 8 + WooPool::INIT_SPACE,
        seeds = [
          WOOPOOL_SEED.as_bytes(),
          wooconfig.key().as_ref(),
          token_mint.key().as_ref(),
          quote_token_mint.key().as_ref()
        ],
        bump)]
    pub woopool: Box<Account<'info, WooPool>>,

    #[account(
        init,
        payer = authority,
        token::mint = token_mint,
        token::authority = woopool
      )]
    pub token_vault: Box<Account<'info, TokenAccount>>,

    #[account(
        has_one = wooconfig,
@>>        has_one = authority,
        has_one = token_mint,
        has_one = quote_token_mint
    )]
    wooracle: Account<'info, Wooracle>,

    #[account(address = token::ID)]
    pub token_program: Program<'info, Token>,
    pub system_program: Program<'info, System>,
}
```

From the above code, it is clear that the permission control for create_pool requires its authority to match wooracle.authority. However, since anyone can create an oracle, an attacker could create an oracle and then create a pool based on that oracle. This breaks the statement made in the README. "Functions need admin authority: claim_fee claim_rebate_fee create_oracle create_pool create_rebate_pool deposit set_pool_admin set_pool_state (all handlers in this file) set_woo_admin set_woo_state(all handlers in this file)"

## Impact

There are two further impacts:

```
1.When the protocol has been running for a period of time, such as 6 months, and
↪  wants to add other token pairs into the swap (e.g., WETH-USDT), the real
↪  administrator may not be able to create the WETH oracle because the WETH
↪  oracle was created by someone else.

2. An attacker could create oracles and pools for other token pairs (e.g.,
↪  WETH-USDT) and set the oracles price and bounds to manipulate prices. Even
↪  if theres a comparison with Pyths prices, it could still pass. As a result,
↪  the attacker could trade with the manipulated prices against pools (e.g.,
↪  USDT, USDC) and steal funds from the pools.
```

## Code Snippet

https://github.com/sherlock-audit/2024-08-woofi-solana-deployment/blob/main/WOOFi_Solana/programs/woofi/src/instructions/admin/create_wooracle.rs#L42

https://github.com/sherlock-audit/2024-08-woofi-solana-deployment/blob/main/WOOFi_Solana/programs/woofi/src/instructions/admin/create_pool.rs#L8

## Tool used

Manual Review

## Recommendation

Set the admin parameter in CreateWooracle to admin = wooconfig.authority.

## Discussion

**toprince**

Impact 1 is valid, it is low impact. Impact 2 is not valid, please verify if it can swap with correct pool.

**sherlock-admin2**

The protocol team fixed this issue in the following PRs/commits: https://github.com/woonetwork/WOOFi_Solana/pull/31

**gjaldon**

Restricting the creation of pools and oracles to only the WooConfig authority fixes the issue.

## Issue M-3: State changes are overwritten during anchor serialization when two accounts are the same

Source: https://github.com/sherlock-audit/2024-08-woofi-solana-deployment-judging/issues/73

### Found by

S3v3ru5

### Summary

The swap operation takes in 3 pool accounts:

1. woopool_from
2. woopool_to
3. woopool_quote

Case 1: In Base-to-Quote Swap, woopool_from is base token pool. woopool_to, woopool_quote will be the quote token pool and both are same accounts `woopool_to == woopool_quote`.

Case 2: In Quote-to-Base Swap, woopool_to is base token pool. woopool_from, woopool_quote will be the quote token pool and both are same accounts `woopool_from == woopool_quote`.

Case 3: In the case of Base-to-Base all three pools will be different.

In case 1 and case 2, two of the passed-in accounts are same. Because of how anchor works, at the end of the execution, changes of only one pool will be saved. For example in case 1 when `woopool_to`, `woopool_quote` are same, at the end of the program execution, updates to either `woopool_to` or `woopool_quote` are recorded.

Anchor `#[derive(Accounts)]` macro works by creating a in-memory copy of the account data, modifying the memory and then writing back into the account data:

- For each of the account, Anchor performs required checks based on the type `Account<'info, WooConfig>`, `Program<'info, Token>`, etc.
- For `Account<'a, T>` type accounts, Anchor deserializes the `AccountInfo.data` into `T`. For e.g, if account X is passed for `Account<'a, WooPool>` then Anchor deserializes `X.data` into type `WooPool` and keeps the copy in memory.
  - Same as `memory` copies of state variables in Solidity.
- The deserialized in-memory copies of the accounts are passed to the instruction-handler: for example `swap` function is an instruction handler.

- After the instruction handler finishes execution, Anchor serializes the accounts and writes into the account data.
- The account data is persistent and hence the state changes are saved.

Consider a Solidity function which copies a `struct` state variable into memory, changes the values in memory and at the end of the function copies the memory values into the state variable. Anchor does the same thing.

This leads to a `storage overwrite` issue when two of the passed-in accounts are same. When the second struct written into account `data`, it will overwrite any changes that are present in the first struct.

The `swap` instruction handler performs the following updates to the pools:

1. Add `from_amount` to `woopool_from` reserve
2. Subtract `to_amount` from `woopool_to` reserve
3. Subtract `swap_fee` from `woopool_quote` reserve
4. Add `swap_fee` to unclaimed fee

https://github.com/sherlock-audit/2024-08-woofi-solana-deployment/blob/main/WOOFi_Solana/programs/woofi/src/instructions/swap.rs#L189-L194

`woopool_quote` is serialized last hence it will overwrite any previous changes if they are same accounts.

- If `woopool_from == woopool_quote`:
  - Update in `1` will be overwritten by `3` and `4`.
  - `from_amount` will not be added to `woopool_from` (quote pool) reserves
- if `woopool_to == woopool_quote`:
  - Update in `2` will be overwritten by `3` and `4`.
  - `to_amount` will not be deducted from the `woopool_to` (quote pool) reserves

## Root Cause

The `swap` function tries to handle all type of swaps in a single instruction allowing for cases where two writable accounts are the same leading to `storage overwrite` issues

https://github.com/sherlock-audit/2024-08-woofi-solana-deployment/blob/main/WOOFi_Solana/programs/woofi/src/instructions/swap.rs#L13-L84

## Internal pre-conditions

*No response*

## External pre-conditions

*No response*

## Attack Path

User performs a swap: either Base to quote or quote to base. The reserves of the quote will not be updated correctly.

## Impact

- If swap sells quote:
  - `from_amount` will not be added to `woopool_from` (quote pool) reserves
- if swap sells base for quote:
  - `to_amount` will not be deducted from the `woopool_to` (quote pool) reserves

This leads to incorrect accounting of `quote pool` reserves variable.

1. The reserve will be higher than the quote pool token vault balance Or
2. The reserve will be lower than the quote pool token vault balance

Additional to incorrect state, because balance is checked before transfering tokens in claim_fee, the issue might prevent admin from claiming fees. The issue might have additional implications that aren't noted here.

## PoC

In the `1_woofi_swap.ts` test file, update the `swap_from_sol_to_usdc` test to print the reserve values of the `toPool` before and after the swap.

Add the following at line 265 in `tests/1_woofi_swap.ts`:

```
let toPoolDataBefore = null;
try {
  toPoolDataBefore = await program.account.wooPool.fetch(toPoolParams.woopool);
} catch (e) {
    console.log(e);
    console.log("fetch failed");
    return;
}
```

```
console.log("toPool reserve before swap:" + toPoolDataBefore.reserve);
console.log("toPool unclaimed fee before swap:" + toPoolDataBefore.unclaimedFee);
```

and Add the following at line 357 at the end of the that test (after `swap` is called':

```
let toPoolDataAfter = null;
try {
  toPoolDataAfter = await program.account.wooPool.fetch(toPoolParams.woopool);
} catch (e) {
    console.log(e);
    console.log("fetch failed");
    return;
}
console.log("toPool reserve after swap:" + toPoolDataAfter.reserve);
console.log("toPool unclaimed fee after swap:" + toPoolDataAfter.unclaimedFee);
```

The output will be:

```
    #swap_between_sol_and_usdc
fromWallet PublicKey:Cybv9pDy9MoysaTk3gkRi3RCtW5gz1jRzZouF47TyfnB
solWalletTokenAccount:GCGi8N26WRcwBedHCFoRo8iycWwLadaNYJx1CW7PyJQ
usdcWalletTokenAccount:Aynux9Ei1FczuPNb5i4Z6cgJisABx3Ty5xNZUfsR6Gst
fromWallet Balance:0
fromTokenAccount amount:1000000
fromTokenAccount decimals:9
toPool reserve before swap:200000
toPool unclaimed fee before swap:0
price - 14597424250
feasible - 1
price - 0
feasible - 0
toAmount:136775
swapFee:4231
toTokenAccount amount:136775
toTokenAccount decimals:6
toPool reserve after swap:195769
toPool unclaimed fee after swap:4231
       swap_from_sol_to_usdc (3274ms)
```

In the test, `woopool_to == woopool_quote == USDC pool`. The `reserve` of the USDC pool should be

```
usdcPool.reserve = usdcPool.reserve - toAmount - swapFee = 200000 - 136775 - 4231
```

However, in the output it can be seen that, the `usdcPool.reserve` is equal to `reserve`

```
- swap_fee = 200000 - 4231 = 195769
```

The deduction of `to_amount` from the `woopool_to.reserve` is not present:

https://github.com/sherlock-audit/2024-08-woofi-solana-deployment/blob/main/WOOFi_Solana/programs/woofi/src/instructions/swap.rs#L190

Only the `swap_fee` deduction is preserved because it was deducted from `woopool_quote`:

https://github.com/sherlock-audit/2024-08-woofi-solana-deployment/blob/main/WOOFi_Solana/programs/woofi/src/instructions/swap.rs#L193

When Anchor serialized the accounts at the end, the `woopool_to` is first written to the account data and then the `woopool_quote`. Because both are same accounts, the changes in `woopool_to` are overwritten when `woopool_quote` is serialized and written to account data.

In case of swap usdc to sol, the changes in `woopool_from` will be overwritten by the `woopool_quote`.

## Mitigation

Divide the `swap` functions into individual `sell_base`, `sell_quote`, and `sell_base_to_base` functions.

## Discussion

**toprince**

Oh....anchor...

**S3v3ru5**

The judging comments include "No loss of funds" as a reason for the medium severity. The `incase_token_got_stuck` can be used to retrieve any tokens from the vaults.

I do not think only loss of funds issues can be consider as High.

The `reserves` variable is a core state variable and it will be wrong because of the above issue. There are multiple implications of the incorrect `reserves` variable

- The issue #68 lists one such impact of this issue.
- The vault token-balance could be greater than reserves when Quote to Base swap is performed. The valid swaps will be rejected even when vault has enough tokens for the swap.
- Pool admin cannot use the `withdraw` function to withdraw the `token balance - reserves` tokens if token-balance is greater than the reserve.

**sherlock-admin2**

The protocol team fixed this issue in the following PRs/commits: https://github.com/woonetwork/WOOFi_Solana/pull/33

**gjaldon**

Case `woopool_from == woopool_quote`:

- This change fixes the issue because only the `woopool_quote` is modified. The changes to the from_pool and the quote pool are applied to only the quote pool because they are the same.

Case `woopool_to == woopool_quote`:

- This change fixes the issue because only `woopool_quote` is modified . The changes to the to_pool and the quote are applied to only the quote pool because they are the same.

Case where all 3 pools are different:

- The previous behavior was retained since this case was unaffected by the issue.

## Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.