Security Assessment Report

Orca Whirlpools

PR768 - Liquidity Locking

February 28, 2025

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the Liquidity Locking PR of the Orca Whirlpools program.

The artifact of the audit was the source code of the following pull request, excluding tests, in https://github.com/orca-so/whirlpools/pull/768.

The initial audit focused on the following versions and revealed 4 issues or questions.

program	type	commit
PR#768 - Liquidity Locking	Solana	26057c46bc614f26aa01792315f3d9a50e565a3d

This report provides a detailed description of the findings and their respective resolutions.

Table of Contents

Result Overview

Issue	Impact	Status
PR#768 - LIQUIDITY LOCKING		
[I-01] Remove the freeze authority when locking positions permanently	Info	Acknowledged
[I-02] Set mint freeze authority if token based locking will be supported	Info	Acknowledged
[Q-01] Are there use cases against freezing position token accounts?	Question	Resolved
[Q-02] Is checking the existence of "lock_config" accounts more versatile?	Question	Resolved

Findings in Detail

PR#768 - LIQUIDITY LOCKING

[I-01] Remove the freeze authority when locking positions permanently

The position account is set as the mint and freeze authorities of the position mint account.

```
/* programs/whirlpool/src/util/token_2022.rs */
094 |     // initialize Mint
095 |     // mint authority: Position account (PDA) (will be removed in the transaction)
096 |     // freeze authority: Position account (PDA) (reserved for future improvements)
097 |     invoke(
098 |         &spl_token_2022::instruction::initialize_mint2(
099 |             token_2022_program.key,
100 |             position_mint.key,
101 |             &authority.key(),
102 |             Some(&authority.key()),
103 |             0,
104 |         )?,
105 |         &[
106 |             position_mint.to_account_info(),
107 |             authority.to_account_info(),
108 |             token_2022_program.to_account_info(),
109 |         ],
110 |     )?;
```

When opening positions, after minting position tokens, the mint authority of the "`position_mint`" is removed, which ensures that each position only has one position token as the authentication token.

```
/* programs/whirlpool/src/util/token_2022.rs */
204 | pub fn mint_position_token_2022_and_remove_authority<'info>(
205 |     position: &Account<'info, Position>,
206 |     position_mint: &Signer<'info>,
210 | ) -> Result<()> {
213 |     // mint
214 |     invoke_signed(
215 |         &spl_token_2022::instruction::mint_to(
216 |             token_2022_program.key,
217 |             position_mint.to_account_info().key,
218 |             position_token_account.to_account_info().key,
219 |             authority.to_account_info().key,
220 |             &[authority.to_account_info().key],
221 |             1,
222 |         )?,
230 |     )?;
```

```
232 |     // remove mint authority
233 |     invoke_signed(
234 |         &spl_token_2022::instruction::set_authority(
235 |             token_2022_program.key,
236 |             position_mint.to_account_info().key,
237 |             Option::None,
238 |             AuthorityType::MintTokens,
239 |             authority.to_account_info().key,
240 |             &[authority.to_account_info().key],
241 |         )?,
248 |     )?;
251 | }
```

Similarly, in the "`lock_position`" instruction, after locking the position token account, the freeze authority of the position mint should ideally be removed. Since the freeze authority cannot be restored once removed, it becomes impossible to unfreeze the position token account thereafter.

Resolution

The team acknowledged this finding and clarified that the current plan is to retain the freeze authority to support potential features such as transferring locked positions under the contract's control. Additionally, since it requires a signature from position PDA, the security remains high unless there is a critical bug in the contract.

PR#768 - LIQUIDITY LOCKING

[I-02] Set mint freeze authority if token based locking will be supported

According to the design decision, the liquidity locking on token program-based positions is not supported. Only token2022-based position mints are supported for this feature.

While the current implementation works for the current design requirements, in the discussions below, it seems token program based position locking may be supported later. If so, the current logic to open positions could use some changes.

In particular, when open a position, "`position_mint.freeze_authority`" is set to "None"

```
/* programs/whirlpool/src/instructions/open_position.rs */
024 | #[account(init,
025 |         payer = funder,
026 |         mint::authority = whirlpool,
027 |         mint::decimals = 0,
028 |     )]
029 |     pub position_mint: Account<'info, Mint>,
```

Anchor generated code to create "`position_mint`" in "`open_position`" is listed as follows:

```
let position_mint: anchor_lang::accounts::account::Account<Mint> = {
let owner_program = AsRef::<AccountInfo>::as_ref(&position_mint)
    .owner;
if !false
    || owner_program
        == &anchor_lang::solana_program::system_program::ID
{
    let __current_lamports = position_mint.lamports();
    if __current_lamports == 0 {
        // system_program::create_account
    } else {
        // system_program::Transfer
        // system_program::Allocate
        // system_program::Assign
    }
    let cpi_program = token_program.to_account_info();
    let accounts = ::anchor_spl::token_interface::InitializeMint2 {
        mint: position_mint.to_account_info(),
    };
    let cpi_ctx = anchor_lang::context::CpiContext::new(
        cpi_program,
        accounts,
    );
    ::anchor_spl::token_interface::initialize_mint2(
        cpi_ctx,
```

```
        0,
        &whirlpool.key(),
        Option::<&anchor_lang::prelude::Pubkey>::None,
    )?;
}
```

According to process_initialize_mint2, the "position_mint.freeze_authority" is set to "None". As a result, "freeze_authority" cannot be set after it's set to "None" (see SetAuthority instruction). It's not feasible to freeze the position token account.

Resolution

The team acknowledged this finding and concluded no action is needed for now to retain the freeze authority for token-based positions.

Currently, only TokenExtensions-based position NFTs are subject to locking. This is because only TokenExtensions-based position NFTs have the Position PDA as their freeze authority. The team has decided to limit the locking mechanism to TokenExtensions-based position NFTs.

Even though locking token-based position NFTs could be supported, the team wants to avoid creating both lockable (newer) and non-lockable (older) positions.

PR#768 - LIQUIDITY LOCKING

[Q-01] Are there use cases against freezing position token accounts?

According to the design description, the goal of position locking is to prohibit operations that decrease position liquidity, while allowing increases in liquidity for locked positions.

However, the design does not specify the desired behavior regarding changes to position token account delegation or position token transfers after locking. Are these actions also prohibited?

The current implementation freezes the position token account when locking a position. After that, the position token account owner cannot approve or revoke delegates. And, the position token cannot be transferred to others.

Since the lock is permanent, are there any requirements to change delegates, given that they can assist with fee collection and other operations? Should the owner or delegate still be allowed to transfer the position token?

Resolution

The team clarified that the demand for transferability does not seem to be very high. And, for liquidity management that involves permanent locking, the need for transfers is even smaller.

If the needs of such transfers become more pressing, instructions like "`transfer_locked_position`" could be added, which is another reason to keep the freeze authority (since it is a PDA, the risk of it being hacked is extremely low compared to a regular wallet).

PR#768 - LIQUIDITY LOCKING

[Q-02] Is checking the existence of "lock_config" accounts more versatile?

Instructions "`decrease_liquidity`", "`decrease_liquidity_v2`", "`lock_position`", and "`close_position_with_token_extensions`" check if the position account is locked before proceeding with their logic.

```
/* programs/whirlpool/src/instructions/decrease_liquidity.rs */
018 | pub fn handler(
019 |     ctx: Context<ModifyLiquidity>,
023 | ) -> Result<()> {
029 |     if is_locked_position(&ctx.accounts.position_token_account) {
030 |         return Err(ErrorCode::OperationNotAllowedOnLockedPosition.into());
031 |     }
```

In particular, it is done by checking if the position token account is frozen or not.

```
/* programs/whirlpool/src/util/shared.rs */
073 | pub fn is_locked_position(
074 |     position_token_account: &InterfaceAccount<'_, TokenAccountInterface>,
075 | ) -> bool {
076 |     position_token_account.is_frozen()
077 | }

/* spl-token-2022-0.9.0/src/state.rs */
115 | pub fn is_frozen(&self) -> bool {
116 |     self.state == AccountState::Frozen
117 | }
```

While the current implementation is correct and serves its purpose, checking for the existence of a corresponding non-empty "`lock_config`" account seems more flexible and deterministic.

This approach preserves the ability to modify the position token's owner and delegate. Additionally, it eliminates concerns about the freeze authority maintenance and the possibility of an unexpected unfreeze.

```
/* programs/whirlpool/src/instructions/lock_position.rs */
012 | pub struct LockPosition<'info> {
018 |     #[account(
019 |         seeds = [b"position".as_ref(), position_mint.key().as_ref()],
020 |         bump,
021 |         has_one = whirlpool,
022 |     )]
```

```
023 |     pub position: Account<'info, Position>,
024 |
035 |     #[account(init,
038 |         seeds = [b"lock_config".as_ref(), position.key().as_ref()],
040 |     )]
041 |     pub lock_config: Box<Account<'info, LockConfig>>,
```

Resolution

Currently, the "`decrease_liquidity`" and similar instructions determine position locking status by checking if the token account is frozen.

While checking for the existence of a "`lock_config`" account would be more direct, the freeze-based approach was chosen intentionally. By leveraging token account frozen status, the backward compatibility with existing implementation was preserved, avoiding modifications to the required accounts.

Appendix: Methodology and Scope of Work

Assisted by the Sec3 Scanner developed in-house, the manual audit particularly focused on the following work items:

- Check common security issues.
- 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

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Sec3 (the "Company") and Orca Management Company, S.A. dba Orca (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

The Sec3 audit team comprises a group of computer science professors, researchers, and industry veterans with extensive experience in smart contract security, program analysis, testing, and formal verification. We are also building automated 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.