Security Assessment Report

Orca Whirlpools

PRs 974, 1010 and 1038

August 22, 2025

Summary

The Sec3 team was engaged to conduct a thorough security analysis of the Orca Whirlpools PRs.

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

The initial audit focused on the following versions and revealed 2 issues or questions.

#	Task	Type	Commit
P1	PR974: Token-2022 support update	solana	6352a9b61a574fb62440a7dca9a933af02847db5
P2	PR1010: Safer account initialization	solana	d5a8bfd83a11de7c1412fe68a2bd42e8f7359c0f
P3	PR1038: Superstate: Non-transferable position	solana	19875ce6595c7e15ad07cd2ede3966b05d34ab62

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

Table of Contents

Result Overview

Issue	Impact	Status
PR974: TOKEN-2022 SUPPORT UPDATE		
[P1-Q-01] Questions on `DefaultAccountState` related checks	Question	Resolved
PR1010: SAFER ACCOUNT INITIALIZATION		
No issues found		
PR1038: SUPERSTATE: NON-TRANSFERABLE POSITION		
[P3-I-01] Migration edge cases	Info	Acknowledged

Findings in Detail

PR974: TOKEN-2022 SUPPORT UPDATE

[P1-Q-01] Questions on `DefaultAccountState` related checks

> *Identified in commit* `6352a9b`.

In this PR, the check for the initialized status of the `DefaultAccountState` token extension has been removed. This means that token badges can now be granted regardless of whether the `DefaultAccountState` is initialized or frozen.

Before:

```
/* programs/whirlpool/src/util/v2/token.rs */
276 | extension::ExtensionType::DefaultAccountState => {
277 |     if !is_token_badge_initialized {
278 |         return Ok(false);
279 |     }
280 |
281 |     // reject if default state is not Initialized even if it has token badge
282 |     let default_state = token_mint_unpacked
283 |         .get_extension::<extension::default_account_state::DefaultAccountState>(
284 |     )?;
285 |     let initialized: u8 = AccountState::Initialized.into();
286 |     if default_state.state != initialized {
287 |         return Ok(false);
288 |     }
```

After:

```
/* programs/whirlpool/src/util/v2/token.rs */
278 | TokenExtensionType::DefaultAccountState => {
279 |     if !is_token_badge_initialized {
280 |         return Ok(false);
281 |     }
282 | }
```

1. Retain the `DefaultAccountState` status check?

While tokens with granted token badges can create liquidity pools, tokens with a frozen `DefaultAccountState` cannot be freely transferred. This seems inconsistent with the core functionality of the Whirlpool protocol. Is it a good idea to retain the `DefaultAccountState` initialization status check?

2. Check the `freeze_authority`?

Additionally, if the token mint's `DefaultAccountState` is `Frozen`, consider checking whether the mint's `freeze_authority` is not `None`. Otherwise, because a mint's freeze authority cannot be re-enabled once disabled, token accounts that are already frozen will never be unfrozen.

Resolution

The team acknowledged the finding in "1. Retain the `DefaultAccountState` status check".

For "2. check the `freeze_authority`", the check has been added in commit `d140c81`.

PR1038: SUPERSTATE: NON-TRANSFERABLE POSITION

[P3-I-01] Migration edge cases

Identified in commit `7f04909`.

The newly introduced `MigrateRepurposeRewardAuthoritySpace` instruction migrates legacy `Whirlpool` accounts to a new version by resetting the `reward_infos[1].extension` and `reward_infos[2].extension` to the default zero values.

```
/* programs/whirlpool/src/instructions/migrate_repurpose_reward_authority_space.rs */
011 | pub fn handler(ctx: Context<MigrateRepurposeRewardAuthoritySpace>) -> Result<()> {
012 |     let whirlpool = &mut ctx.accounts.whirlpool;
014 |     // Check if the whirlpool has already been migrated
016 |     // Notes: Whirlpool accounts with reward_infos[2].authority equal to [0u8; 32]
017 |     // do NOT exist on the four networks where the Whirlpool program is deployed.
018 |     if whirlpool.reward_infos[2].extension == [0u8; 32] {
019 |         panic!("Whirlpool has been migrated already");
020 |     }
022 |     // Migrate the reward authority space
023 |     whirlpool.reward_infos[1].extension = [0u8; 32];
024 |     whirlpool.reward_infos[2].extension = [0u8; 32];
026 |     Ok(())
027 | }
```

The migration uses `reward_infos[2].extension == 0` as a flag to determine if a `Whirlpool` account has already been migrated. This is based on the assumption that **all** existing whirlpool `reward_infos[2].extension` hold non-zero values as mentioned by the comments in lines 16-17.

However, the `SetRewardAuthority` instruction allows manual zero-setting of `reward_infos[2].extension` without resetting `reward_infos[1].extension`. If this behavior occurred before the deployment of this PR, the zero-value check may fail to detect that a migration is still necessary.

```
/* programs/whirlpool/src/instructions/set_reward_authority.rs */
007 | pub struct SetRewardAuthority<'info> {
008 |     #[account(mut)]
009 |     pub whirlpool: Account<'info, Whirlpool>,
010 |
011 |     #[account(address = whirlpool.reward_infos[reward_index as usize].authority)]
012 |     pub reward_authority: Signer<'info>,
013 |
014 |     /// CHECK: safe, the account that will be new authority can be arbitrary
015 |     pub new_reward_authority: UncheckedAccount<'info>,
016 | }
017 |
018 | pub fn handler(ctx: Context<SetRewardAuthority>, reward_index: u8) -> Result<()> {
019 |     ctx.accounts.whirlpool.update_reward_authority(
020 |         reward_index as usize,
021 |         ctx.accounts.new_reward_authority.key(),
022 |     )
```

```
023 | }
```

Furthermore, after the deployment of this PR, the `SetRewardAuthorityBySuperAuthority` instruction is restricted to updating only `reward_infos[0].extension`, which means the `SuperAuthority` cannot fix this issue.

As a result, `reward_infos[1].extension` may remain stuck in an inconsistent or invalid state.

```
/* programs/whirlpool/src/instructions/set_reward_authority_by_super_authority.rs */
022 | pub fn handler(ctx: Context<SetRewardAuthorityBySuperAuthority>, _reward_index: u8) -> Result<()> {
023 |     ctx.accounts
024 |         .whirlpool
025 |         .update_reward_authority(ctx.accounts.new_reward_authority.key())
026 | }
027 |

/* programs/whirlpool/src/state/whirlpool.rs */
209 | /// Update the reward authority.
210 | pub fn update_reward_authority(&mut self, authority: Pubkey) -> Result<()> {
211 |     self.reward_infos[0]
212 |         .extension
213 |         .copy_from_slice(&authority.to_bytes());
214 |
215 |     Ok(())
216 | }
```

Therefore, although the comment mentions that Whirlpools with `reward_infos[2].authority == [0u8; 32]` do not exist, since it's allowed by the code, we'd like to confirm if this holds before migration.

Otherwise, before deploying this PR, consider manually setting all whirlpool `reward_infos[2].extension` to non-zero through team intervention.

Resolution

The team acknowledged this finding. After checking four networks where the Whirlpool program has been deployed, the team has not found any data in a conflicted state so far.

The team understands the risk if someone sets the reward authority to `Pubkey::default` before the PR is deployed and considers the risk minimal. In the worst case, if a conflicting `set_reward_authority` call were made, the team could still update the `migrate_repurpose_reward_authority_space` instruction to handle it.

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 (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.