By
Quit

Contract Review
Issue date
05/2/2025 (updated 5/5/2025)

Overview

The following is a review of Roach Racing's Trax Redemption contract.

Contracts in scope for this review include:

- `contracts/TraxRedeem.sol`

This review is based on SHA `02f90521afbc05761936113a8e4992fd2ac72638`, and aims to identify security vulnerabilities, opportunities for gas optimization, and general best practice recommendations with regards to the contracts in scope. The review should not be considered an endorsement of the project, nor is it a guarantee of security.

A subsequent review of remediations was conducted using SHA `4d18de5dea1a093eac3a989e6bad563a98e8517b`.

Findings

G-01: Unnecessary validation of burned amount in `redeem`
Severity: gas optimization
Status: Acknowledged
The redeem function contains safety checks to ensure that the amount burned matches the expectation. When dealing with the known Trax contract, there is no path to burning the wrong amount when using `useFrom`. Removing the comparison allows for removal of two `balanceOf` calls, and some basic arithmetic.

G-02: Unnecessary use of `ReentrancyGuard`
Severity: Gas Optimization
Status: Resolved (`35ae4f17ba1aec8bea67b32ae62213dc81c7aa61`)
The `nonReentrant` modifier is used in three locations: `redeem()`, `withdrawAll()`, and `withdraw()`. withdrawAll makes only one external call, `USDC.transfer()`. This hands execution over to the USDC contract, but that in turn does not hand execution over to anything else. `withdraw()` makes several external calls, but again they are all to the USDC contract (sometimes passing through `TraxExchange`). Finally, `redeem()` also makes several external calls, but just like `withdraw()` they are all to the USDC contract (or `TraxExchange`). Recommend removing `ReentrancyGuard`.

G-03: `_sendTokens` contains duplicate reserves check

Severity: Gas Optimization
Status: Resolved (`35ae4f17ba1aec8bea67b32ae62213dc81c7aa61`)
The internal `_sendTokens` function is only called in one place, from within `redeem`. `getAvailableBalance` is used to determine if there are sufficient USDC reserves, however, redeem begins by checking this same condition via the `enoughReserves()` function. Recommend removing the duplicate check.

L-01: Centralization Risk
Severity: Low
Status: Acknowledged
There are two methods by which an authorized address can effectively disable redemptions for USDC.

- An address with `DEFAULT_ADMIN_ROLE` can withdraw all USDC from `TraxRedeem`, which can lead to a `LowReserves()` state which will cause a revert during any redemption attempts.
- An address with `SET_PRICE_ROLE` on `TraxExchange` can use `setPrice()` to cause a mismatch in USDC price between `TraxExchange` and `TraxRedeem`, which will result in a `PriceChanged()` error during any redemption attempts.

This is a low severity issue given the inherent trust assumptions with such a system design.

L-02: Signature Misuse
Severity: Low
Status: Resolved (`4d18de5dea1a093eac3a989e6bad563a98e8517b`)
The Trax contract's `useFrom()` passes `msg.sender` as the account param for the internal `_use` function. This then gets used to validate the signature and emit the `Used` event, meaning the signature must be signed with the `TraxRedeem` contract as the account and there is no parameter tying it to the user. A user could use a signature intended for another user to burn their Trax and redeem it for USDC. There is no frontrunning on Abstract (sequencing is done strictly on a FCFS basis), so an attacker would have to obtain a valid signature by other means. Furthermore, the attacker would not be stealing any Trax from the user as the burn still deducts from their own account. Team has opted to use the `param` component of the signature to denote the user whose funds are being spent.

Summary

Overall, there are no major issues found with `TraxRedeem`. Several gas optimizations can be made with relatively little code changes, so those are recommended before deploying to Abstract. However, the code will function as stands.