By
Quit

Contract Review
Issue date
01/26/2025

Overview

The following is a review of Roach Racing, an upcoming trading sim/racing hybrid game launching on Abstract. The platform pits racers against each other on a track, with their PNL from simulated longing or shorting an asset adds or detracts from their speed.

Contracts in scope for this review include:

- `contracts/TRAX.sol`
- `contracts/TraxExchange.sol`
- `contracts/interfaces/ITRAX.sol`

This review is based on SHA `68c006578e45dec46edfa97212669ac3e7523ff4`, 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.

Findings

I-01: `minLimitPerTx` should be named `mintLimitPerTx`
Severity: Informational
Status: Resolved (`140ff923e4ca4116a701e78817554b148eba99b0`)
This is just a typo on Trax.sol.

I-02: Missing Index for Event Parameters
Severity: Informational
Status: Resolved (`140ff923e4ca4116a701e78817554b148eba99b0`)
The `Used` event in the TRAX contract emits an id parameter that is used to track token burns and link them to specific orders. However, this parameter is not marked as indexed, making it difficult to efficiently query these events. The same goes for the `paymentToken` parameter of the `Price` and `Exchange` events in `TraxExchange`.

Recommend adding the `indexed` keyword to the `Used.id`, `Price.paymentToken` and `Exchange.paymentToken` parameters.

I-03: Lack of signature validation library
Severity: Informational
Status: Acknowledged

A signature requirement was added in order to validate the ID passed to `use` and `useFrom`. This uses raw `ecrecover` accesses. Recommend using Solady's `SignatureCheckerLib` zkSync variant for this purpose.

G-01: Use of error strings
Severity: Gas Optimization
Status: Resolved (`140ff923e4ca4116a701e78817554b148eba99b0`)
Recommend using custom errors to revert, as strings in Solidity are very expensive. This will save a bit of gas at deploy, as well as each time a revert is hit.

G-02: traxToken can be made immutable
Severity: Gas Optimization
Status: Resolved (`140ff923e4ca4116a701e78817554b148eba99b0`)
`traxToken` is set at construction time and does not have a setter for reassigning it later. Save an SLOAD on every read of `traxToken` by making the token immutable. This will also mean renaming the variable to uppercase `TRAX_TOKEN` according to the Solidity Style Guide.

L-01: Lack of Zero Address Validation in Constructor
Severity: Low
Status: Partially Resolved
The constructor accepts address parameters for `defaultAdmin`, `setPriceRole`, and `withdrawRole` without validating that they are not the zero address. Additionally, the `_traxToken` parameter is not validated.

Recommend adding zero address validation in the constructor.

Update: `defaultAdmin` is now validated in the constructor. Since this role can assign other roles, the others do not strictly require validation.

L-02: Unreliable Order ID System in Token Burning Functions
Severity: Low
Status: Resolved (`55507d006bcdf14ae8e0f0074b77cab192489b9b`)
The `use()` and `useFrom()` functions in the TRAX contract allow users to burn tokens while specifying an arbitrary id parameter. This ID is intended to link token burns to specific orders in an off-chain database, but the current implementation allows users to specify any ID value when burning tokens. The backend assigns IDs to users before the transaction is submitted, so there is no risk of theft here. However, the potential for ID reuse is messy and should be addressed.

Recommend using a backend signer to sign for valid orders, and passing the signature to the `use()` and `useFrom()` functions to ensure IDs passed are as expected. The functions should mark the `ID` as used and revert if they are submitted again.

Summary

The TRAX Exchange contract implements a straightforward token exchange system with role-based administrative controls. The contract is generally well-structured and implements basic security controls through OpenZeppelin's AccessControl. While the core smart contract functionality appears sound, there are some minor suggestions that should be implemented before deployment.

During the review, we were also asked to examine the key management system for contract signers. This revealed security concerns around the storage of sensitive credentials in GitLab CI/CD variables. While not directly related to the smart contract implementation, we strongly recommend migrating all secrets to AWS Secrets Manager and replacing direct AWS credentials with IAM roles. The tx-manager service should be configured to use an IAM instance role with minimal necessary permissions to access these secrets and KMS for decryption. This would eliminate GitLab as a single point of failure, provide better audit logging, and enable proper key rotation practices.

Though it was not included in the review, it's worth noting that the `TraxExchange` contract implements nonstandard decimal handling, which could lead to user confusion. We recommend standardizing all token amount handling to use full decimal precision, consistent with ERC20 practices. Not a security risk as the current implementation will work as expected, moreso a QOL suggestion here.

While none of the identified issues are critical, addressing them would improve the contract's usability and maintainability. The key management improvements, while outside the scope of the smart contract itself, should be prioritized to ensure the overall security of the system.