2025-12-04 Decentraland Name Registrar Executor Report

Date: 2025-12-04
Author: Facundo Spagnuolo
Commit: 1b0319e53d0554ba987c8bbdb4621ad434476abe
Amendment: 8654df202976649ba91f35c30126239e175208e6

Introduction

The `RegisterNameCrossChainExecutor` is a specialized contract that facilitates cross-chain name registrations within the Decentraland ecosystem. It acts as a restricted execution module that validates and processes name registration transactions through the Coral bridge contract, enforcing fee limits and execution constraints.

This contract is not intended to be called directly by end users. Instead, it is designed to be invoked exclusively by the `CreditsManagerPolygon` contract, which handles user authentication, signature verification, credit accounting, and replay protection. Through this architecture, users are able to register Decentraland names on Ethereum mainnet while paying with MANA credits on Polygon, with the executor operating under strictly controlled and predefined execution assumptions.

Assumptions

A1. Name prices are considered constant

Name prices are set to a constant value of 100 MANA, as stated in the contract (see L24). If a new price is required, the `RegisterNameCrossChainExecutor` contract can be paused, and a new executor contract will need to be deployed.

A2. Decetraland will cover the underlying bridge fees

The `RegisterNameCrossChainExecutor` contract is designed to use the Coral bridge to execute cross-chain name registrations on the destination chain. This process requires the payment of bridge fees denominated in MANA (see L130-L135). These fees are intended to be subsidized by Decentraland, rather than charged to the end user initiating the name registration. Accordingly, the contract is expected to maintain a sufficient MANA balance in advance to cover such fees.

A3. Decentraland securely builds Coral bridge execution calls

The `RegisterNameCrossChainExecutor` contract is intended to be used exclusively through the `CreditsManagerPolygon` contract, which ensures that execution requests are authorized and validated by a trusted signer (see L619-L625). Consequently, although the executor contract does not independently validate the inner Coral call data, it relies on the credits manager to prevent arbitrary or malicious call construction.

A4. Reply attacks are already covered by the `CreditsManagerPolygon` contract

Replay attacks against the Coral execution flow are effectively prevented by the `CreditsManagerPolygon` contract, which enforces strict uniqueness and authorization guarantees before invoking the `RegisterNameCrossChainExecutor` (see L609-L614).

A5. Coral bridge execution is considered safe and correct

The `RegisterNameCrossChainExecutor` contract relies on the Coral bridge to execute cross-chain name registrations on the destination chain (see L139-L144). It is assumed that the Coral bridge behaves as expected and that the invoked `fundAndRunMulticall` execution path correctly and deterministically results in the requested name registration when provided with valid parameters.

Findings

Critical

None

High

None

Medium

M1. Avoid using a generic struct in favor of explicit params

The `ExternalCall` struct aggregates multiple semantically independent inputs (target address, function selector, calldata, fee metadata, and expiration) into a single generic container. While flexible, this design reduces clarity and makes it harder to reason about the exact invariants being enforced by the contract.

Consider using explicit function parameters instead of a generic struct to improve readability, reduce the risk of misconfiguration, and make validation logic more explicit and auditable. In particular, it would help ensure that each parameter is individually constrained and validated, rather than relying on implicit assumptions about how the struct is constructed.

Additionally, explicit parameters reduce the surface for unintended behavior, and they make it easier for external reviewers to understand which inputs are trusted, derived, or user-controlled.

Update: Partially addressed at 7081bdaaea1c69c86d8c5b20dd24da56375d721f.

M2. Consider making Coral dependency not immutable

The `coral` address is defined as an immutable dependency, meaning it cannot be updated after deployment. While this provides deployment-time certainty, it also introduces rigidity in case the Coral contract needs to be updated.

Consider making the Coral dependency configurable to increase operational flexibility and reduce the risk of the executor becoming unusable or requiring redeployment in scenarios where the Coral contract changes.

If configurability is introduced, it should be protected by strict access control (such as `DEFAULT_ADMIN_ROLE`), include event emission for transparency, and potentially leverage a timelock or governance process to mitigate the risk of misconfiguration or abuse.

Update: Won't fix. Decentraland team states that a new implementation can be deployed in case the Coral dependency changes.

M3. No invariant check to ensure Coral consumes exactly what's expected

The `RegisterNameCrossChainExecutor` contract approves the Coral contract to spend an amount equal to `NAME_PRICE + manaFee` (see L134-L135), but it does not enforce any invariant verifying how much MANA is actually consumed during execution. As a result, the transaction may succeed even if Coral consumes less than the approved amount or does not consume the expected funds at all.

While this design relies on the correctness and honesty of the Coral integration, the absence of a post-execution invariant makes it harder to reason about correctness and accounting guarantees. For example, partial execution paths could result in leftover allowances or unspent balances without being detected by the executor.

Consider introducing a post-execution check to verify the contract's MANA balance before and after execution, or asserting that the allowance has been fully consumed, to strengthen correctness guarantees by ensuring that the execution only succeeds if Coral actually consumes the intended `NAME_PRICE + manaFee`. This would also make failures more explicit and easier to diagnose.

Update: Won't fix.

Low

L1. Errors on Coral calls are not bubbled up

The contract executes the Coral call using a low-level `call` and only checks the returned success flag. In the event of a failure, the transaction reverts with a generic `CallFailed` error, discarding the revert reason and any return data provided by the Coral contract (see L139-L144).

While this pattern prevents silent failures, it limits debuggability and makes it harder for integrators, off-chain services, and operators to diagnose the root cause of execution issues. Relevant context, such as specific revert reasons, custom errors, or protocol-level failures originating from the Coral contracts, are not propagated to the caller or observable on-chain.

Consider bubbling up the revert data to improve transparency and facilitate troubleshooting, without materially increasing the trust or attack surface of the contract.

Update: Addressed at 8654df202976649ba91f35c30126239e175208e6.

L2. MANA fee balance is not verified to be held by the contract

The `RegisterNameCrossChainExecutor` contract validates that the requested MANA fee does not exceed a configured USD-denominated maximum (see L130-L132). However, it does not explicitly verify that the contract holds a sufficient MANA balance to cover the approved fee amount at the time of execution.

As a result, execution relies on the implicit assumption that the contract is pre-funded with enough MANA to cover bridge and routing fees. If this assumption does not hold, the Coral call may fail at execution time, leading to a reverted transaction after allowances have been set and external calls have been attempted.

While this does not directly introduce a security vulnerability, consider adding an explicit balance check to improve robustness, prevent avoidable failures, and make the contract's operational assumptions more explicit.

Update: Won't fix.

L3. `maxUSDMANAFee` naming is confusing

The variable name `maxUSDMANAFee` conflates both the asset (MANA) and the denomination (USD), which makes its actual meaning harder to immediately understand. Despite the name, the value represents a maximum fee expressed in USD, later converted to MANA using an on-chain price feed, not a maximum MANA amount.

Consider renaming this variable to something more explicit, such as `maxUsdFee` or `maxFeeUsd` to improve readability and reduce cognitive overhead for reviewers and developers. The same applies to related internal functions and events, whose names currently propagate this ambiguity.

Update: Addressed at 7081bdaaea1c69c86d8c5b20dd24da56375d721f.

L4. Unify authorization control logic

The `execute` function currently enforces authorization by directly checking whether the caller equals the `creditsManager` address (see L130-L132). While effective, this approach diverges from the role-based access control mechanism (`AccessControl`) used throughout the rest of the contract.

Unifying authorization logic by introducing a dedicated role (e.g., `EXECUTOR_ROLE`) and granting it to the `creditsManager` would improve consistency, readability, and extensibility. A role-based approach would also make it easier to support future changes, such as rotating the credits manager, adding additional authorized executors, or integrating governance or timelock mechanisms, without requiring contract redeployment.

Consider aligning all authorization checks under the same `AccessControl` module to reduce the risk of misconfiguration, simplify auditing, and make the contract's permission structure more explicit and easier to reason about.

Update: Won't fix. Decentraland team states this permission is required to be immutable; therefore, it's preferred not to use `AccessControl` for this.

L5. Roles should represent permissions, not identities

The `RegisterNameCrossChainExecutor` contract currently mixes role-based permissions with implicit identity-based authorization. For example, while a `PAUSER_ROLE` is defined, the `pause` function additionally allows the `DEFAULT_ADMIN_ROLE` to bypass this role check and pause the contract (see L193-L197).

In a role-based access control model, roles are intended to explicitly represent permissions rather than identities or hierarchical authority. Under this paradigm, any account that is allowed to pause the contract should be granted the `PAUSER_ROLE`, and administrative accounts should not implicitly inherit operational permissions unless explicitly assigned.

Update: Won't fix.

Notes

N1. Avoid hardcoding the Coral function selector

The `RegisterNameCrossChainExecutor` contract hardcodes the function selector (`0x58181a80`) corresponding to `fundAndRunMulticall`, rather than deriving it from the function signature at compile time (see L118-L121). While functional, this approach reduces readability and increases the risk of errors or inconsistencies if the Coral interface changes or if the selector is misdocumented.

Consider using a symbolic reference like `ICoral.fundAndRunMulticall.selector` to make it clearer, improve maintainability, and provide compile-time guarantees that the selector matches the intended function signature. Avoiding hardcoded selectors also reduces the risk of subtle bugs introduced during refactors and makes the code easier to review.

Update: Addressed at 7081bdaaea1c69c86d8c5b20dd24da56375d721f.

N2. Inconsistent documentation for `_maxUSDMANAFee` decimals

The `RegisterNameCrossChainExecutor` constructor documentation states that the `_maxUSDMANAFee` parameter is expressed with 8 decimals (see L78), while the rest of the contract logic treats this value as having 18 decimals.

Consider updating the documentation to accurately reflect the expected decimal format to improve correctness and avoid misconfiguration.

Update: Addressed at 7081bdaaea1c69c86d8c5b20dd24da56375d721f.

N3. Comment that the MANA/USD rate is safe to cast to `uint256`

The `_validateMANAFee` function casts the MANA/USD rate returned by the price aggregator from `int256` to `uint256` (see L159-L161). While unchecked casts from signed to unsigned integers can be error-prone, in this case, the conversion is safe because the base `AggregatorHelper` contract already enforces that the returned rate is strictly positive (see L20-L24). However, this safety property is implicit and not documented at the call site.

Consider adding a local comment to clarify that the value is guaranteed to be non-negative by prior validation and improve the code clarity.

Update: Won't fix.

N4. Rename `CallFailed` to `ExecutionFailed`

The custom error `CallFailed` is used to signal a failure during the execution of the external Coral call. However, the error name is relatively generic and may be confused with other types of failed calls within the contract or broader system.

Consider renaming this error to `ExecutionFailed` which is aligned with `ExecutionExpired` and will make it clearer that the failure happened during the execution phase.

Update: The error was finally removed at 8654df202976649ba91f35c30126239e175208e6.

N5. Unify naming convention

The `RegisterNameCrossChainExecutor` contract uses inconsistent naming conventions for similar concepts, particularly around the use of casing and word order for "MANA", "USD", and "fee". For example, events such as `ManaUsdAggregatorUpdated` and `MaxUSDMANAFeeUpdated`, as well as the error `MANAforFeeExceeded`, follow different capitalization and composition patterns despite referring to closely related fee and pricing concepts.

These inconsistencies reduce readability and make the codebase harder to maintain. Consider adopting a consistent naming convention (e.g., `ManaUsd`, `MaxUsdFee`, `ManaFeeExceeded`) to improve

clarity and reduce cognitive overhead for developers, auditors, and integrators.

Update: Addressed at 0de1d241bd0fadb4659e68e76483275ef20318ae.

N6. Follow Solidity recommended function ordering

The `RegisterNameCrossChainExecutor` contract does not consistently follow the recommended Solidity function ordering pattern, as described in the official Solidity style guide. The current layout intermixes these in a way that deviates from standard expectations.

Consider refactoring the contract to conform to the recommended ordering to support a better development experience and maintainability.

Update: Addressed at 7081bdaaea1c69c86d8c5b20dd24da56375d721f.