RegisterNameCrossChainExecutor Smart Contract Audit Report

Commit: 1b0319e53d0554ba987c8bbdb4621ad434476abe

Scope: RegisterNameCrossChainExecutor.sol

Repo: https://github.com/decentraland/offchain-marketplace-contract

Auditor: https://github.com/fzavalia

Fixes: beac17d8d67923af410abf838a920d2ae63e991e

Summary

The **RegisterNameCrossChainExecutor** contract enables users to purchase **Decentraland Names**, which are native to **Ethereum mainnet**, using **MANA credits** directly from **Polygon**.

Before the contract can be used, both the contract address and its `execute` selector must be authorized by an address holding the **DEFAULT_ADMIN_ROLE** in the **CreditsManagerPolygon** contract. This requirement exists because any contract outside the official Decentraland Marketplaces must be explicitly approved by an admin to accept credits.

When a user receives a credit signed by a trusted off-chain signer and wants to use it to purchase a name through the **RegisterNameCrossChainExecutor**, they require an additional custom external call. This call, which contains the necessary execution data, must also be signed by a trusted signer.

With both signatures, the user calls the `useCredits` function in **CreditsManager** and provides the custom call. This triggers an interaction with the **RegisterNameCrossChainExecutor**, which then communicates with the **Coral** contract to initiate the cross-chain purchase.

Internally, the **CreditsManager** authorizes the **RegisterNameCrossChainExecutor** to utilize the credited MANA. The executor pulls NAME_PRICE (100 MANA) from the **CreditsManager** and approves the **Coral** contract to access both the name price and bridge fees when calling `fundAndRunMulticall`.

Finally, the **CreditsManager** calculates how much MANA was consumed in the external transaction and deducts that amount from the user's credit.

It is important to note that **bridge fees** are paid using **MANA** held by the executor contract itself. The user only pays with their **MANA credits** to cover the **name price** (100 MANA), while the **fees** required for the cross-chain transaction are funded directly by the executor contract's balance.

Assumptions

A1 - The custom external call signer is trusted

The external call signer is secured and under control of the Decentraland team. A compromised external call signer could result in the executor being used for purposes other than those intended.

A2 - The Coral contract interaction is trusted

The Coral contract pulls the MANA (name price) and bridge fees from the executor contract to perform the cross-chain transaction.

It is assumed that the Coral contract interaction will use the provided calldata to buy the name on mainnet and transfer it to the user.

It is also assumed that since the data is not validated on-chain, the signer of the custom external call is making those validations externally.

A3 - Users don't pay for bridge fees

The line `mana.transferFrom(creditsManager, address(this), NAME_PRICE);` only pulls the MANA for the name price (100 MANA), while an additional `manaFee` is being approved for the Coral contract. This means the executor contract must hold MANA in its balance to pay for bridge fees. If users are expected to pay for these fees, they should be pulled from the credits manager as well.

A4 - The MANA fee provided in the calldata is computed off-chain

Given that the fee might increase between the quote and the transaction execution, ensure that the MANA fee provided off-chain includes a buffer percentage to avoid transaction reverts due to minor bridge fee increases (even by 1 wei).

Findings

Critical

-

High

-

Medium

-

Low

L1 - NatSpec documentation decimal inconsistency for maxUSDMANAFee

The constructor NatSpec at line 81 incorrectly states `_maxUSDMANAFee` is "in 8 decimals", but the calculation at line 174 and other documentation correctly indicate it should be "in 18 decimals". Update the constructor documentation to match the correct 18-decimal format.

ADDRESSED

Also changed the name of the _maxUSDMANAFee to _maxFeeUSD

L2 - Unnecessary data in RegisterNameCrossChainExecutor's ExternalCall

The executor contract has a struct that is received by the execute function with the following schema:

```
struct ExternalCall {
    address target;
    bytes4 selector;
    bytes data;
    bytes extra;
    uint256 expiresAt;
}
```

This external call is encoded into the `data` field of the CreditsManager's ExternalCall:

```
struct ExternalCall {
    address target;
    bytes4 selector;
    bytes data;
    uint256 expiresAt;
    bytes32 salt;
}
```

The executor's struct could be simplified to:

```
struct ExternalCall {
    bytes data;
    uint256 manaFee;
}
```

Rationale:

- `target` can be removed because the executor only calls the Coral contract, which is immutable
- `selector` can be removed because only the `fundAndRunMulticall` selector is valid
- `expiresAt` is unnecessary because expiration is already validated by the CreditsManager in its own struct's `expiresAt` field
- `extra` is replaced by `manaFee` to avoid re-decoding

The target, selector, and expiresAt validations can be removed, which will save gas and reduce complexity.

ADDRESSED

Removed selector and expiresAt from struct. No more selector and expiresAt validations. Data now includes the whole function call data instead of only the parameters. Mana fee still decoded from extra param.

UPDATE

The selector doesn't matter anymore, since the data of the struct has to have the selector to be called. The signer could sign a call to the coral contract with any selector. Not a problem given that the signer is trusted. But might cause trouble if the signer is compromised and other selectors are used (which might not be the case depending on the coral contract).

Informational

I1 - Use AccessControlEnumerable instead of AccessControl

Using `AccessControlEnumerable` would enable easy querying of addresses that have been given a particular role via `getRoleMemberCount(bytes32 role)` and `getRoleMember(bytes32 role, uint256 index)` .

With the current `AccessControl` , querying role addresses requires knowing the addresses beforehand and calling `hasRole(address account, bytes32 role)` , which necessitates relying on an indexer, checking events on Etherscan, or manually tracking role addresses.

WAIVED

I2 - manaUsdAggregator and manaUsdAggregatorTolerance could be made immutable

Given that the MANA/USD price feed from Chainlink will be used, and the tolerance will likely be set to the heartbeat (27 seconds), the probability of these parameters needing to be changed is negligible. Making them immutable would reduce user gas costs by eliminating 2 SLOAD operations.

ADDRESSED

Also removed unnecessary update functions for the now immutable variables.

I3 - Consider providing the pauser role on deployment

Providing the pauser role during deployment would reduce deployment steps. Instead of the current process (deploy with EOA → set pauser role → transfer ownership to MultiSig → remove admin role from EOA), the contract could be deployed directly with the MultiSig as owner and the pauser role assigned.

WAIVED

I4 - withdrawERC721 function may be unnecessary

There appears to be no valid reason for this function to exist. The `withdrawERC20` function can be used to withdraw MANA from the contract for various admin purposes, but the contract never holds NFTs except by accident. Note that accidental NFT transfers would require a normal transfer, as `safeTransfer` would fail due to the contract lacking an `onERC721Received` function.

WAIVED

I5 - Event indexing improvements

Several events do not index the sender of the transaction, which could be useful for off-chain filtering: `Executed`, `ManaUsdAggregatorUpdated`, and `MaxUSDMANAFeeUpdated`.

Additionally, the `Executed` event could benefit from indexing a hash of the external call to make it easier to query specific execution transactions.

The contract also doesn't track or emit the actual MANA consumed by Coral (only the approved amount). Consider emitting the actual MANA consumed by comparing balances before and after the Coral call for better monitoring of fee estimates versus actual consumption.

WAIVED

I6 - Consider moving the forceApprove after the transferFrom

It would be more conventional to approve funds after the contract has pulled them from the credits manager rather than before.

ADDRESSED