OpenZeppelin | security

CreditsManagerPolygon Audit

April 29, 2025

Table of Contents

Summary

Type	Contracts
Timeline	From 2025-04-10 To 2025-04-15
Languages	Solidity

Total Issues	8 (2 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (0 resolved)
Low Severity Issues	1 (1 resolved)
Notes & Additional Information	5 (1 resolved)

Scope

OpenZeppelin audited the decentraland/offchain-marketplace-contract repository at commit 4fb7e3d.

In scope was the following file:

```
/src/credits/
└── CreditsManagerPolygon.sol
```

System Overview

Decentraland is introducing a new smart contract that would allow users to utilize pre-signed credits for marketplace transactions on the Polygon network. The `CreditsManagerPolygon` contract enhances the user experience by facilitating off-chain credit issuance that can be redeemed on-chain, while also significantly reducing the necessity for direct MANA transfers, streamlining the entire transaction process.

Users can earn credits through various means within the ecosystem and then use these credits to purchase Decentraland assets either directly from creators in the primary market or from other users in the secondary market. To access each of these markets, the `CreditsManagerPolygon` contract integrates with the Decentraland Legacy Marketplace, which is the current Marketplace implementation, and the Collection Store. Both of these marketplaces have specific functions that are callable by the contract. However, arbitrary external calls can also be allowed when carefully selected and approved by the contract owner. This is to ensure adaptability for any emerging use cases.

Trust Assumptions and Privileged Roles

During the audit, multiple trust assumptions under which the system operates were identified:

- All the external and out-of-scope contracts that have not been audited as part of this engagement behave in a manner that is correct and in the best interest of the protocol and its users.
- All the external and out-of-scope contracts are free of exploits and function as intended.
- The privileged roles would behave in a rational and non-malicious way that preserves the integrity of the protocol.

Multiple privileged roles were identified in the `CreditsManagerPolygon` contract:

- `DEFAULT_ADMIN_ROLE`: Can grant/revoke roles and perform administrative functions.

- `CREDITS_SIGNER_ROLE`: Can sign credits that users can later redeem within the contract.
- `PAUSER_ROLE`: Can pause the contract functionality.
- `USER_DENIER_ROLE`: Can restrict certain users from utilizing credits.
- `CREDITS_REVOKER_ROLE`: Can revoke credits that have been previously issued.
- `EXTERNAL_CALL_SIGNER_ROLE`: Can authorize specific external calls.
- `EXTERNAL_CALL_REVOKER_ROLE`: Can revoke any custom external call signature.

Medium Severity

M-01 MANA Balance Check in `useCredits` Can Be Bypassed

The `useCredits` function of the `CreditsManagerPolygon` contract verifies that the caller's MANA balance remains unchanged before and after the external call. As noted in the comments, this check aims to prevent users from using credits to effectively acquire `MANA` by purchasing their own assets.

However, this check can be bypassed if the external call interacts with the `DecentralandMarketPlace` contract. Specifically, during the call to the marketplace's `accept`/`acceptWithCoupon` function, the NFT is transferred to the address specified in the trade (`sent.beneficiary`) using `IERC721.safeTransferFrom`. If the recipient is a contract, the `onERC721Received` hook is triggered. A malicious contract can exploit this hook to manipulate its MANA balance during the transfer of the NFT, circumventing the balance check. A similar issue can also arise when the external call is made to the `CollectionStore` contract if the item being transferred implements a `_beforeTokenTransfer` hook.

Although on-chain identity is not a foolproof method (since the same entity can use many different addresses), to prevent the same address from being both the buyer and the seller when it calls the `CreditsManagerPolygon` contract, consider explicitly comparing the caller of the `useCredits` function and the trade's beneficiaries since this check is more robust than the balance check.

Update: *Acknowledged, not resolved*

M-02 Caller Context Affects Merkle Proof Validation in Market Place Integration

The `accept` and `acceptWithCoupon` functions of the `DecentralandMarketPlace` contract include checks to validate a trade before accepting it. One of these checks verifies that if a non-empty `allowedRoot` is provided, the `allowedProof` is a valid Merkle proof, showing that the caller is included in the seller's list of allowed addresses.

However, if the `accept` or `acceptWithCoupon` function is called indirectly through the `CreditsManagerPolygon` contract, the caller in the context of the `DecentralandMarketPlace` will be the `CreditsManagerPolygon` contract and not the actual buyer. As a result, it is not possible for the marketplace to effectively check if the actual buyer is in the list allowed by the seller addresses.

Consider validating the Merkle proof in the `CreditsManagerPolygon` contract before calling the marketplace's `accept` / `acceptWithCoupon` function.

Update: *Acknowledged, not resolved*

Low Severity

L-01 Missing Multiple Array Length Check

When a function has two or more correlated array parameters, it is crucial to check that these arrays have the same length to prevent data inconsistencies or partial execution. The `denyUsers` function accepts multiple array parameters without verifying their length equality.

Ensure that all functions with multiple correlated array parameters including a check to verify that these arrays have the same length before processing them.

Update: *Resolved in pull request **#21** at **commit f2216a8**.*

Notes & Additional Information

N-01 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract

incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

The `CreditsManagerPolygon` contract does not have a security contact.

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Acknowledged, not resolved*

N-02 Missing Named Parameters in Mappings

Since Solidity 0.8.18, developers can utilize named parameters in mappings. This means mappings can take the form of `mapping(KeyType KeyName? => ValueType ValueName?)`. This updated syntax provides a more transparent representation of a mapping's purpose.

Within `CreditsManagerPolygon.sol`, multiple instances of mappings without named parameters were identified:

- The `isDenied` state variable
- The `isRevoked` state variable
- The `spentValue` state variable
- The `allowedCustomExternalCalls` state variable
- The `usedCustomExternalCallSignature` state variable

Consider adding named parameters to mappings in order to improve the readability and maintainability of the codebase.

Update: *Acknowledged, not resolved*

N-03 Emitting Update Events Without Value Changes

When a setter function does not check if the value being set is different from the existing one, it becomes possible to set the same value repeatedly, leading to event spamming. While the following functions can be only called by addresses holding designated roles, they nonetheless emit an update event upon execution, regardless of whether the new value differs from the

current one. This can potentially confuse off-chain clients monitoring the events for state updates.

In the `CreditsManagerPolygon.sol` contract, multiple instances of possible event spamming were identified:

- The `denyUsers` function
- The `revokeCreditSignatures` function
- The `updateMaxManaCreditedPerHour` function
- The `updatePrimarySalesAllowed` function
- The `updateSecondarySalesAllowed` function

Consider adding a check that reverts the transaction if the value being set is the same as the existing one.

Update: *Acknowledged, not resolved*

N-04 Improve Credit Tracking By Using Message Hash Instead of Signature Hash

To spend a credit, a user must present a valid ECDSA signature from an address holding the `CREDITS_SIGNER_ROLE`. The signed message consists of the hash of the following tuple: `(sender, chainID, creditManagerAddress, credit)`. Credits are not necessarily spent in full at once. To track partial usage, the `CreditsManagerPolygon` contract maintains a mapping called `spentValue` which records the amount of each credit that has been spent. The current implementation uses the hash of the signature as the key for this mapping.

However, due to the non-deterministic nature of ECDSA, multiple valid signatures can be generated for the same message using different randomness. This means that, if the same credit has been signed multiple times, resulting in different signatures, and, therefore, different keys in the `spentValue` mapping, this would allow potential double-spending of the same credit. The Decentraland team informed the audit that, in practice, each credit is expected to be signed only once. While this assumption may hold operationally, it introduces a risk if not enforced at the contract level.

Consider using the hash of the message as the key in the `spentValue` mapping instead of using the hash of the signature. This ensures consistent tracking of credit usage and removes reliance on off-chain assumptions.

Update: *Resolved in pull request **#21** at commit **45a4fd3**.*

N-05 Early Asset-Type Reversion for Gas Optimization

The `useCredits` function relies on the `_handleMarketplacePreExecution` function, which performs necessary checks before making an external call to the `MarketPlace`. One of these checks involves verifying the asset type that will be sent by the seller and, therefore, received by the caller of the `useCredits` function.

If the asset type is `ASSET_TYPE_ERC721`, `_handleMarketplacePreExecution` checks whether secondary sales are allowed in the `CreditsManagerPolygon` contract. If the asset type is `ASSET_TYPE_COLLECTION_ITEM`, it checks whether primary sales are allowed. However, if the asset type does not match either of these, the function allows execution to continue, only to revert later in the external `MarketPlace` call, since only these two asset types are permitted.

To optimize for gas usage, consider reverting early in `_handleMarketplacePreExecution` if the asset type does not match any of the expected ones.

Update: *Acknowledged, not resolved*

Conclusion

The `CreditsManagerPolygon` contract adds support for users to execute orders using the credits they earned in exchange for Decentraland assets in either primary or secondary markets. Serving as a bridge that enables off-chain credit issuance to be consumed on-chain, reducing the need for direct MANA transfers from user wallets.

The audit team was able to uncover multiple medium-severity issues within the codebase. Overall, the code quality was found to be respectable, and the Decentraland team is appreciated for being prompt in answering questions and sharing documentation.