Decentraland Security Analysis

by Pessimistic

This report is public

November 10, 2021

Abstract 2
Disclaimer 2
Summary 2
General recommendations 2
Project overview 3
  Project description 3
  Code base update 3
Procedure 4
Manual analysis 5
  Critical issues 5
  Medium severity issues 5
    Overpowered owner 5
    No documentation (fixed) 5
  Low severity issues 6
    Code quality 6

Abstract

In this report, we consider the security of smarts contracts of Decentraland project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

Summary

In this report, we considered the security of Decentraland smart contracts. We performed our audit according to the procedure described below.

The initial audit showed an Overpowered owner issue of medium severity. Also, a few low severity issues were found.

The project has a documentation, the code base is covered with NatSpec comments.

After the initial audit, the code base was updated. CEI pattern violation issue was fixed with this update.

General recommendations

We recommend addressing an overpowered owner issue and implementing CI to run tests, calculate code coverage, and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Decentraland project on a public GitHub repository, commit 37114d6477a7728882129da1fd142e801ec412fd.

The scope of the audit only included **registries/ThirdPartyRegistry.sol** and **managers/Tiers.sol** files with their dependencies.

The project has no documentation, and compiles with warnings. However, the code base has detailed NatSpec comments.

All tests pass, the code coverage is 100%.

The total LOC of audited sources is 540.

Code base update

After the initial auidt, the code base was updated. For the recheck, we were provided with commit 4e6f5198af72445d0fdddbe3dcf9ff7670bb511f.

In this update, one issue was fixed.

Also, the documentation for the project was provided with this update.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan project's code base with automated tools: Slither and SmartCheck.
  - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
  - We manually analyze code base for security vulnerabilities.
  - We assess overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They often lead to the loss of funds or other catastrophic failures. The contracts should not be deployed before these issues are fixed. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

Overpowered owner

The owner can change prices for tiers, change tokens, and set arbitrary fee collector. As a result, the owner can front-run transactions of other users or buy slots at a lower price.

In the current implementation, the system depends heavily on the owner. Therefore, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if the owner's private keys become compromised.

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., multisig.

No documentation (fixed)

The project has no documentation. As a result, it is sometimes unclear what the intention of the code is, and whether its behavior is correct, and the architecture of the project is appropriate.

Considering the complexity of the project, the documentation is critically important not only for the audit but also for development process. It should explicitly explain the purpose and behavior of the contracts, their interactions, and main design choices.

The documentation for the project was provided.

Low severity issues

Low severity issues do not directly affect project's operations. However, they might lead to various problems in the future versions of the code. We recommend taking them into account.

Code quality

- CEI pattern (checks-effects-interactions) is violated in **ThirdPartyRegistry** contract, e.g., in `buyItemSlots()` function. We highly recommend following CEI pattern since it improves usage predictability.

  The issue has been fixed and is not present in the latest version of the code.

- Storage variables are `internal` by default. Consider declaring visibility of `nonces` mapping explicitly at line 21 of **NativeMetaTransaction** contract to improve code readability.

- In **NativeMetaTransaction** contract, consider declaring parameters `userAddress` and `relayerAddress` of `MetaTransactionExecuted` event at lines 17–18 as `indexed`.

- In **NativeMetaTransaction** contract, consider using `ECDSA` library from OpenZeppelin instead of `ecrecover()` to protects against signature malleability attacks at line 95.

- Names for local variables should not duplicate names of functions or storage variables to avoid shadowing. Consider renaming the `_owner` variable in:
  - **Tiers** contract at lines 26 and 31.
  - **ThirdPartyRegistry** contract at line 101.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Daria Korepanova, Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

November 10, 2021