Orb Land Upgradability And Registry Audit Report

July 15 2023

Prepared for Orb.land by NFR Audits
nofrontrunning@gmail.com

NFR AUDITS

Summary

This is a follow-up audit on the Orb Land's newly implemented upgradability and invocation registry. This update introduced upgradability to the Orb, Orb Pond, and Invocation Registry, as well as some necessary refactors and function updates. The upgradability design is based on the UUPS pattern with OpenZeppelin's implementation. The Orb contract can be upgraded sequentially based on its current version by the owner and/or keeper depending on the Orb's holding status. The upgradability for the Orb Pond and Invocation Registry is controlled by the official Orb Land.

The audit scope covers 4 files from 1 GitHub repository with commit hash [63fa48516636a1f2633b6459ed8f82c5233dda7d].

- Orb.sol
- OrbPond.sol
- OrbInvocationRegistery.sol
- CustomUUPSUpgradeable.sol

Codebase Overview

The codebase is neat and clean, the system is well-designed and well-engineered. All functions are properly commented following the NatSpec format. It has been a pleasure to audit this codebase.

Audit Methods

The codebase is audited manually line by line. Auto detectors are also used to ensure best coverage of vulnerabilities.

Trust Assumptions

- The Orb Land's official account has certain controls over the upgradability functions for some contracts, such as the Orb Pond and Invocation Registry's implementation, Orb implementation version addresses. We assume the official account acts in a trustworthy manner.
- Certain deployment and initialization processes should be handled with care to ensure no front running occurs, such as in the case of the Orb Pond. We assume that these processes are carried out correctly.
- The `authorizedContracts` mapping in the Invocation Registry should only allow non-malicious contracts. We assume that this is handled correctly by the Orb Land's official account.

Severity Classification

Severity classification in this report uses Immunefi Vulnerability Severity Classification System v2.2 for smart contracts.

Critical Issues

No critical issues were found during the audit.

High Issues

No high issues were found during the audit.

Medium Issues

Medium 01 - Orb upgradability could be broken if incorrect implementation is used

The Orb function `upgradeToNextVersion()` handles the UUPS upgradability for the Orb contract, it does this by calling function `_upgradeToAndCall()` in line 1082 inherited from OpenZeppelin's `ERC1967UpgradeUpgradeable` contract.

However this function does not check the rollback safety of proposed new implementation, the Orb's upgradability could be broken if a non-compatible implementation has been used.

Recommended Fix : Use `_upgradeToAndCallUUPS()` function from ERC1967UpgradeUpgradeable contract instead, which implements the safety rollback check.

Severity partially mapped to Level 3 Medium - Smart contract unable to operate due to lack of token funds, not due to lack of funds but mis-use of function.

Update: This issue is now fixed at PR 145

Low Issues

Low 01 - Supporting unofficial Orbs in OrbInvocationRegistry could lead to storage pollution

Current design of `OrbInvocationRegistry` does not check if the calling Orb contract is deployed by the Orb Land, it simply allows any contract implementing the IOrb interface to interact and update storage in the registry.

Although no immediate threats were found, this design could lead to undesired storage pollution from third party Orbs and leave a potential risky attack surface for the future. Plus, there are already incentives to deploy third party Orbs (for example to avoid certain cost), supporting them via the offical registry might further incentivise the usage of unofficial Orbs.

Recommended Fix : Restrict the official registry to only work with Orbs deployed by the Orb Land.

Severity partially mapped to Level 3 Medium - Griefing, marked as low due to low damage.

Update: After discussing with the client, they decided to not fix this issue due to reasons below. We reviewed this decision and agreed this is suitable for the current implementation, the Orb Pond can also be upgraded when necessary to address this issue.

> *OrbPond does not have an efficient way of checking Orb origin. There's an array of Orbs, but it's not designed to check if the Orb was issued by this pond. Otherwise, any Orb can set any value as their pond. So I'd need to add isOrbFromSelf mapping address->bool to OrbPond, to check that it is so. I'm*

> *leaning towards not fixing this and allowing storage polution. We haven't seen anyone deploy unofficial Orbs yet.*

Low 02 - Lack of reserved storage gaps for upgradable contracts

Multiple upgradable contracts throughout the codebase are designed to be inheritable, e.g. the Orb, OrbPondand OrbInvocationRegistry . However these contracts do not reserve any storage gaps, this could lead to potential storage collision in the inheriting contract if parent contracts are upgraded with new storage variables in the future or restrict the ability to add new storage variables into these contracts.

Recommended Fix : Reserve some storage gaps in these upgradable contracts.

Severity partially mapped to Level 2 Low - Contract fails to deliver promised returns, but doesn't lose value, contracts might fail due to incorrect upgrades.

Update: This issue is now fixed at PR 146, storage gaps are added to all three contracts.

Low 03 - lack of input checks when registering new Orb versions in the Pond could temporarily break Orb's upgradability

The Orb can only be upgraded sequentially based on its current version to an non-zero implementation address, however the function `registerVersion` in the Orb Pond does not enforce any checks if there is any version gaps with implementation `address(0)` in the version path, which could be caused by registering a new version that's more than 1 version higher than current latest version or updating a previous version to `address(0)` . This could temporarily break the Orb's upgradability until the issue is fixed by the admin.

Recommended Fix : Add checks to ensure the version path remain valid after any updates, e.g.

- Do not allow updating a lower version implementation to `address(0)`
- Do not allow setting new implementation address for versions higher than latest version + 1
- Decreasing the latest version after un-registering the latest version

Severity partially mapped to Level 2 Low - Contract fails to deliver promised returns, but doesn't lose value, contract might fail to operate correctly due to misusage. However the impact is low since the admin could easily correct the mistake.

Update: This issue is now fixed at PR 147 where new checks have been added to ensure the version path is valid.

Low 04 - Missing event emissions

Some functions in the codebase lack event emissions after storage updates, for example

- function `authorizeContract()` in OrbInvocationRegistry
- function `registerVersion()` in OrbPond

Recommended Fix : Emit an event whenever storage variables are updated to ensure easy off-chain filtering, monitoring and state reconstruction when needed.

Severity partially mapped to Level 1 Low - Contract fails to deliver promised returns, but doesn't lose value, contract fails to emit events for storage updates.

Update: This issue is now fixed at PR 148, events are added to these functions.

Notes

Note 01 - Typos

There are a few typos in the code comments, for example

- "It is the only Orb state variable that can needs to be written by..." in the comment of function `setLastInvocationTime()` in line 1015 of the Orb contract should be "can be" or "needs to be".
- "Returns the version of the Orb" in the comment of function `version()` in line 125 of the OrbPond contract should be "Returns the version of the Orb Pond".

Recommended Fix : Correct these typos.

Severity mapped to Level 1 None - Best practices

Update: This issue is now fixed at PR 149

Note 02 - Unnecessary double inheritance

There are a few cases of double inheritance in the codebase, e.g. In the Orb contract, `IERC165Upgradeable` is already inherited by `ERC165Upgradeable` in line 77, there is no need to double inherit it.

Other similar cases

- `IERC721Upgradeable` in the Orb contract which is already inherited by `IERC721MetadataUpgradeable` .
- `Initializable` in the Orb, OrbPond and OrbInvocationRegistry contract is already inherited by `UUPSUpgradeable` .

Recommended Fix : Duplicated inheritance is not a big issue, some times we do allow them just to be explicit. However if this is not intended, consider removing these duplicated inheritance.

Severity mapped to Level 1 None - Best practices

Update: This issue is now fixed at PR 150

Note 03 - Modified Open Zeppelin contract used

The Orb uses a modified version of Open Zeppelin's UUPSUpgradable contract to tailor to the 2-step upgrade process of the Orb. While the reason is understandable and no issues was found in this contract during the audit, it is still not a recommended way of using Open Zeppelin's library as stated on their website.

```
To keep your system secure, you should always use the installed code as-is, and
neither copy-paste it from online sources, nor modify it yourself.
```

Recommended Fix : Use overrides instead of direct modifying Open Zeppelin's contract library.

Severity mapped to Level 1 None - Best practices

Update: After discussing with the client, they decided to not fix this issue due to reasons below. We reviewed this decision and agreed this modification does not raise any concerns at this stage.

> *This one is another difficult one. Instead of using CustomUUPSUpgradeable, I could inherit from UUPSUpgradeable and revert in both public methods... We do this with ERC721 methods because we want interface-level compatibility, but there's nothing like that for upgrades. I tested implementing that: it costs us 900 bytes (~14% of margin) for no benefit. So thinking about not implementing this suggestion.*