Orb Land Audit Report

May 29 2023

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

NFR AUDITS

Orb Land Audit Report

Summary

The Orb is an immutable, non-freely-transferable ERC 721 NFT that allows the holder to ask the creator a question regularly while imposing a Harberger tax on the holder. The current Orb holder sets a purchase price at which anyone can buy the Orb from the current holder. However, the holder also pays a Harberger tax based on this price. All parties pre-deposit Eth into the Orb to participate. The Orb allows withdrawal of excess amounts and foreclosing on the Orb if the holder is insolvent.

The audit scope covers 2 files from 1 GitHub repository with commit hash d86ba6401bb67d5995473b5a32a4214813ce1fea.

- Orb.sol
- IOrb.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 codebase is immutable and permissionless without a central entity. Orb creators can set their Orb's parameters as they wish. However current official Orbs are designed to be deployed by the Orb Land team, we assume this process is done correctly.

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

High 01 - Malicious Orb creators can steal from holders by front running the purchase transaction when purchasing the Orb from creator

The Orb allows the creator to set a few critical parameters when it has not been purchased yet, e.g.

- Harberger tax rate
- Royalty fee
- Cool down period
- Clear text length

However, a malicious Orb creator can front run any purchase transactions to update these parameters in order to steal from or exploit the Orb buyer, e.g.,

- Setting the tax rate very high and call `settle()` shortly after purchase to steal all holder's deposit
- Setting the royalty fee to `FEE_DENOMINATOR - 1` to effectively steal all future sales income
- Setting the cool down to `COOLDOWN_MAXIMUM_DURATION` to minimize invocation times, which might not be what the buyer agreed to when purchasing
- Setting the clear text length to 1 to stop the holder from invoking the Orb with clear text

This attack requires the Orb to be held by the creator when the purchase transaction happens, this can be easily achieved through multiple ways like `listWithPrice()` if the Orb is held by itself, or direct purchase if held by a third party.

Although it's challenging to enforce all the Orb's functions if the creator becomes malicious, it's still a good idea to eliminate obvious risks from a design perspective to ensure fair rules for both parties involved.

Recommended Fix : Include all sensitive parameters in `purchase()` and check these has not been updated.

Severity mapped to Level 5 Critical - Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield. Reported as High due to the rare condition that requires the Orb creator to be malicious.

Update : This issue is now fixed in PR 121 commit 755bbe3073d94e36a2555aeff299cebfb03603f3 where all sensitive information are now checked at purchase time together with a unified event to handle any mismatch values.

Medium Issues

Medium 01 - Flag response function could be abused

The Orb allows holders to express their dissatisfaction with the creator's answers through flagResponse(). This is a rather powerful tool could indicate the quality of the responses and impact future purchases. However there are a few potential issues,

1. Mistakenly flagged responses cannot be corrected. The system is designed without a de-flagging function to discourage the holder from using it to threaten the creator; however, this could still happen before flagging.
2. If a holder loses his Orb due to a purchase after a response, and then re-purchases the Orb back within the flag cooldown time requirement, they are still unable to flag the response due to the `holderReceiveTime` being reset during the latest purchase.

3. creators can front run any flagging transactions with a purchase transaction to avoid been flagged. This will cause the flag transaction to fail due to `onlyHolder` modifier.
4. A holder could potentially avoid high Harberger tax by setting the price low, making all responses private, and flagging them all to indicate a low quality of responses, which could discourage future buyers from purchasing the Orb.

Designing a system with potential on-chain dispute resolution can be difficult without making it overly complicated. However, certain fixes might be implemented to mitigate some potential attacks.

Recommended Fix :

- Adding a de-flag function could solve issue 1.
- Associating responses with holder at the time instead of relying on current holder and `holderReceiveTime` could allow flagging even if the current holder has changed. This could solve issues 2 and 3.
- Issue 4 is relatively hard to solve, depending on how complicated the system is allowed to be, potential solutions might be a forced frozen state for resolution, human intervention for mediation etc.

Severity mapped to Level 3 Medium - Griefing and unauthorized access.

Update : After discussing with the client, they decided to not fix this issue due to a few reasons quoted below. We agreed this approach is reasonable and suitable for current use cases of the Orb.

> *Overall, flagging primarily concerns the social layer, and has a lot of considerations how to manage that off-chain.*

> *We chose to implement tracking invoker addresses on chain for separate reasons, unrelated to flagging, in PR #127.*

> *With regards to flagging design, no changes will be made, but some implications of the system will be documented. Flagging rules remain somewhat simple: only the current, solvent holder can flag, only responses received since the last purchase, and only for the duration equal to Orb cooldown. No other exceptions to these rules exist.*

> *Documented implications of these rules:*

> - *Invoker might not be able to flag the response, if it arrives after the Orb has been purchased from the invoker. This is to limit another type of griefing attack: invoker no longer has anything at stake and could flag the response just to tank the price. The same reasoning is why we require holder solvency to flag a response.*

> - *Invoker will not be able to flag the response even if they are the current Orb holder, if the Orb has been purchased from them, then the response was delivered, and original invoker re-purchased the Orb. First, we consider the original invoker to be satisfied just from the Orb purchase. Additionally, re-buying the Orb means wanting to actually own and use the Orb, not to just report feedback via flagging. We allow the invoker to always read the response, even if it's private. So it means the original invoker chose to re-buy the Orb knowing the response contents. It's far more likely the re-bought because the response was good and they had use for the Orb, and far less likely that they spent considerable amount just out of spite, to flag the response.*

Low Issues

Low 01 - Creator can fork Orb to avoid potential platform fees

The Orb has an immutable beneficiary address set in the constructor to receive all creator fees including initial auction fee, royalty fee and Harberger tax. The comment in the codebase also indicates a platform fee to be implemented through a fee splitter at `beneficiary` address.

```
The intended use case for the beneficiary is to set it to a revenue splitting
contract. //line 67
```

However, an Orb creator can simply fork the Orb code and deploy their own version to avoid any platform fees. The creator has the option and incentive to redirect current holder and potential buyers to use the newly forked version. The only hassle might be the lack of a front end, but universal front end solutions from Etherscan and Monobase could be used instead.

It is generally hard to stop fee eliminating forks for an open-sourced project, the current plan to not support forked Orbs in the frontend is a reasonable, however weak mitigation due to the alternatives listed above. Plus, there is no easy way to verify an Orb's origin on-chain even if users want to.

Recommended Fix : Consider using a factory deployer to deploy, manage, index all official Orbs, enforce platform fees, and restrict non-official Orb's interactions with official contracts (potential future functions). Although this solution does not eliminate forking of the Orb, it does allow any frontends and the potential users to easily verify an Orb's origin on-chain and discourage forks.

Severity mapped to Level 4 High - Theft of unclaimed royalties, reported as Low due to rare condition of creator forking the Orb.

Update : This issue is now fixed in PR 122 commit 2604c844fc54fd6ffe6270c491558f136a2426ef where a new `OrbPond` is added as the factory deployer to index all deployed orbs as well as handle deploying parameters.

Low 02 - Beneficiary withdraw function might not withdraw all beneficiary's funds

Public function withdrawAllForBeneficiary() is designed to allow the beneficiary to withdraw all funds. However, during this process, the `_withdraw()` function does not call `_settle()` to update the tax revenue for the beneficiary. Although it is possible to manually call `settle()` before `withdrawAllForBeneficiary()` , it is not an ideal solution due to the two transactions are likely to span over different blocks and causing the beneficiary's withdraw balance to be out of date.

Recommended Fix : call `_settle()` for `beneficiary` in function `_withdraw()` or `withdrawAllForBeneficiary()` .

Severity mapped to Level 2 Low - Contract fails to deliver promised returns, but doesn't lose value.

Update : This issue is now fixed in PR 123 commit 13997356ed7877ab80ced10b657126f8d105224f where `_settle()` is added into `withdrawAllForBeneficiary()` .

Low 03 Weak price manipulation mitigation when purchasing Orb

The purchase() function has a price manipulation mitigation to check lastSettlementTime >= block.timestamp in line 743 to prevent the following potential attacks

1. Front-running `purchase()` , e.g. attacker could set the price to 0 and front-run all `purchase()` transactions with `setPrice()` to 1 or 0 wei alternately to avoid both Harberger tax and Orb been purchased.
2. Composed transaction with price manipulation immediately before purchasing to avoid royalty fees to achieve free transfers.

While it works well against the first attack, the mitigation is relatively weak against the second one because it only allows public intervention of the attack surface for a minimum of only one block (roughly 12 seconds) if the attacker attacks over a span of two blocks.

Recommended Fix : Increase the time frame of the check to reduce the attack surface and allow more time for public intervention e.g. purchasing the Orb at a low price.

Severity mapped to Level 4 High - Theft of unclaimed royalties. Reported as Low due to the mitigation already in place and a relatively complicated attack vector.

Update : The client decided to not fix this issue due to the reason quoted below. We think this is a reasonable approach for current use cases of the Orb.

> *more discussions lead to a decision not to address this weakness, and automate punishment of cheating Orb holders with bots.*

Notes

Note 01 - Gas optimization

The codebase sets initial values to 0 for variables with a default 0 value in multiple places. These cost extra gas and are not necessary. e.g.,

```
uint256 public auctionStartingPrice = 0; // line 119
uint256 public invocationCount = 0; // line 161
uint256 public flaggedResponsesCount = 0; //line 167
```

Recommended Improvement : Removing value assigning, use default values instead.

Severity mapped to Level 1 None - Best practices.

Update: this issue is now fixed in PR 124 commit dc59dbe8aab44cc78084587e5e36f31facc0d7c2 where all three values have been removed.

Note 02 - Redundant codes

There are a few redundant codes,

- Function `finalizeAuction()` sets price to 0 in line 550 if there is no bidders. However, at this stage, the price is already 0 due to the Orb being held by the Orb contract itself, which sets the Orb price to 0 during minting, relinquishing, and foreclosure. There is no need to set the price to 0 again.

- `purchase()` sets lastSettlementTime to current block time in line 768. However `lastSettlementTime` is already set to the block time during `_settle()` in line 677, which is called by `purchase()` in line 737, no need to set it again.

Recommended Improvement : Remove redundant codes.

Severity mapped to Level 1 None - Best practices.

Update: The issue is now fixed in PR 125 commit 95e8301cda0a694e10824e39217584b9c490830b and commit 90e55c513124f180a70dfbb6c015e0bfb4ced7d2. The client spotted an issue in the recommendation, since `_settle()` could return early if owner is the current holder, it is still necessary to set the `lastSettlementTime` which makes the second recommendation invalid. However after discussing with the client, we agreed the best solution is to adopt the recommendation and update `_settle()` to set `lastSettlementTime` even if owner is the current holder, this refactor solves this issue and still keep the codebase clean.

Note 03 - Unused named return variables

There are multiple unused named return variables in the codebase, named return variables are an alternative to direct return value and are only needed if used in the function body. e.g.

- The `isInterfaceSupported` return variable in the `supportsInterface` function.
- The `baseURIValue` return variable in the `_baseURI` function.
- The `isAuctionRunning` return variable in the `auctionRunning` function.
- The `auctionMinimumBid` return variable in the `minimumBid` function.
- The `isHolderSolvent` return variable in the `holderSolvent` function.
- The `feeDenominatorValue` return variable in the `feeDenominator` function.
- The `holderTaxPeriodSeconds` return variable in the `holderTaxPeriod` function.
- The `owedValue` return variable in the `_owedSinceLastSettlement` function.
- The `isResponseFound` return variable in the `_responseExists` function.

Recommended Improvement : Removing unnecessary named return variables.

Severity mapped to Level 1 None - Best practices.

Update: The client decided to not make any changes due to the reason quoted below. We think the decision makes sense and does not cause any security concerns.

> *While the named return values are not used in the implementation, they are used when generating documentation, and they don't give any overhead in compiled bytecode.*

Note 04 - Some events lack indexed parameters

Some events defined in `IOrb.sol` has no indexed parameters, e.g.

- event Creation in line 11
- event AuctionStart in line 14
- event AuctionExtension in line 16
- event PriceUpdate in line 25
- event OathSwearing in line 39
- event HonoredUntilUpdate in line 40
- event AuctionParametersUpdate in line 41
- event FeesUpdate in line 51
- event CooldownUpdate in line 57
- event CleartextMaximumLengthUpdate in line 58

Recommended Improvement : Index these events for off-chain filtering.

Severity mapped to Level 1 None - Best practices.

Update: This issue is now fixed in PR 128 commit b1c099c018f7a5a875414cdeeff939a2451c9d7a where these events are now indexed.