



# Neura Security Assessment

Oct 30, 2023

Prepared for

**Reezo**

Prepared by:

**0xfoobar**

# Table of Contents

Table of Contents	2
Summary	3
Overview	3
Project Scope	3
Summary of Findings	3
Disclaimer	4
Key Findings and Recommendations	5
1. Rarity reveal can be manipulated with MEV	5
Description	5
Impact	5
Recommendation	5
Response	5
2. Reentrancy attack in admin ETH withdrawal	6
Description	6
Impact	6
Recommendation	6
Response	6
3. msg.value used in internal functions	7
Description	7
Impact	7
Recommendation	7
Response	7
4. Use of transfer to send ETH can cause reverts	8
Description	8
Impact	8
Recommendation	8
Response	8
5. Use latest compiler version and associated features	9
Description	9
Impact	9
Recommendation	9
6. Use named mappings	10
Description	10
Impact	10
Recommendation	10
7. Use block.chainid instead of embedded assembly	11
Description	11
Impact	11

Recommendation	11
8. Use custom errors instead of require strings	12
Description	12
Impact	12
Recommendation	12
9. Remove pragma abicoder v2	13
Description	13
Impact	13
Recommendation	13
10. Fix typos	14
Description	14
Impact	14
Recommendation	14
11. Standardize msg.sender and _msgSender() usage	15
Description	15
Impact	15
Recommendation	15
12. Standardize license usage	16
Description	16
Impact	16
Recommendation	16
13. Replace private mappings + public getters with public mappings	17
Description	17
Impact	17
Recommendation	17

# Summary

## Overview

Neura is a novel NFT project.

## Project Scope

We reviewed the core smart contracts and test suite contained within commit hash 548795205481d75cd5f69dc4b72fedf4bf89de78 at <https://github.com/ReezoETH/neura>. Fixes were reviewed at commit hash 7e66cc2e6aa56285a9d2d85a2f9ee4ccbaa81384 at <https://github.com/Adanede/neura>.

## Severity Classification

High - Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).

Medium - Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.

Low - Assets are not at risk: state handling, function incorrect as to spec, issues with comments.

Informational - Code style, clarity, syntax, versioning, off-chain monitoring (events, etc)

## Summary of Findings

Severity	Findings	Resolved
<b>High</b>	<b>0</b>	<b>-</b>
<b>Medium</b>	<b>2</b>	<b>1</b>
<b>Low</b>	<b>1</b>	<b>0</b>
<b>Informational</b>	<b>10</b>	<b>3</b>



# Disclaimer

This security assessment should not be used as investment advice.

We do not provide any guarantees on eliminating all possible security issues. foostudio recommends proceeding with several other independent audits and a public bug bounty program to ensure smart contract security.

We did not assess the following areas that were outside the scope of this engagement:

- Website frontend components
- Offchain order management infrastructure
- Multisig or EOA private key custody
- Metadata generation

# Key Findings and Recommendations

## 1. Rarity reveal can be manipulated with MEV

Severity: **Medium**

Files: Android.sol

### Description

The contract method `reveal()` is public and sets both rarity and the specific gene with input randomness of `block.timestamp`, `block.number`, `blockhash(block.number)`, and a nonce. While these variables are tough to predict in advance, a determined attacker could wrap the reveal in an outer atomic smart contract check or take advantage of private mempool transaction bundling to even perform this revert from a seemingly innocuous EOA call. Only commit-reveal schemes or an admin-gated reveal functions can protect against this.

### Impact

Technical users could mint a disproportionate number of rares for themselves.

### Recommendation

Consider using a commit-reveal scheme, admin-gated reveal, or other async sources of randomness that can thwart randomness manipulation attacks.

### Response

The androids differ in skill but not rarity. We want to keep the onchain mechanics because it's such a key part of the collection. The market can determine the rarities and value over the long run. Because we implemented choosing of android type for users...no pseudorandom is left.

## 2. Reentrancy attack in admin ETH withdrawal

Severity: **Medium**

Files: Module.sol

### Description

The contract method `withdraw()` splits ETH between two admin addresses based on parameter percentage splits. However the split calculation is not frozen; the second address receives whatever remaining balance exists after paying the first address.

### Impact

First admin could steal all funds with a reentrancy triggered in its fallback or receive function. This would repeatedly loop through the pro rata calculation taking parts of address2's share each time.

### Recommendation

Calculate `transfer2Amount` right after `transfer1Amount`, then transfer that frozen calculation to address2. This way reentrancy attacks would fail if the second address doesn't receive their full share.

### Response

The attack is theoretically possible, but not practically in current conditions.



### 3. msg.value used in internal functions

Severity: **Informational**

Files: PreSale.sol, Module.sol

#### Description

External payable functions do not have immediate value checks, but pass on msg.value to internal functions which revert if not enough eth has been paid.

#### Impact

This is dangerous because even if the immediate code may be safe, it's trivial to slip up and either double-check or miss a check when msg.value is being relayed around internal functions.

#### Recommendation

Only query msg.value once in direct external functions, then pass along its amount as a parameter to internal functions that should know about it.

#### Response

Acknowledged, will keep this flow because the current contract is safe as-is.

## 4. Use of transfer to send ETH can cause reverts

Severity: **Low**

Files: AndroidLootBox.sol, Module.sol, PreSale.sol

### Description

The protocol employs Solidity's `.transfer` method to send ETH. However, `.transfer` is limited to a hardcoded gas amount of 2300, which can break if contracts have fallback logic. This is also potentially incompatible with future changes to Ethereum's gas costs.

### Impact

Sending ETH to contracts with fallback logic will revert.

### Recommendation

Use the `.call()` method as described in <https://solidity-by-example.org/sending-ether/>.

### Response

Not fixed.

## 5. Use latest compiler version and associated features

Severity: **Informational**

Files: hardhat.config.ts, \*.sol

### Description

All files currently have the solidity pragma at 0.8.11. Best practice is to target the latest 0.8.21 version. This will have three benefits: cheaper deployment costs, gas optimization and new language features. This protocol is targeting Ethereum, so there are no PUSH0 incompatibility issues. However if multichain expansion is a plan in the future, it'll be important to explicitly set the compiler's target EVM version as "paris" rather than "shanghai" to avoid PUSH0 opcode incompatibility.

### Impact

Increased deployment size and gas costs.

### Recommendation

Use a floating ^0.8.21 pragma everywhere, and target the paris EVM version with `evm\_version = "paris"` in foundry.toml

### Response

Contract pragmas are now ^0.8.0. Deployment scripts are not included but the latest could be used. To support custom errors a min pragma of ^0.8.4 is recommended.

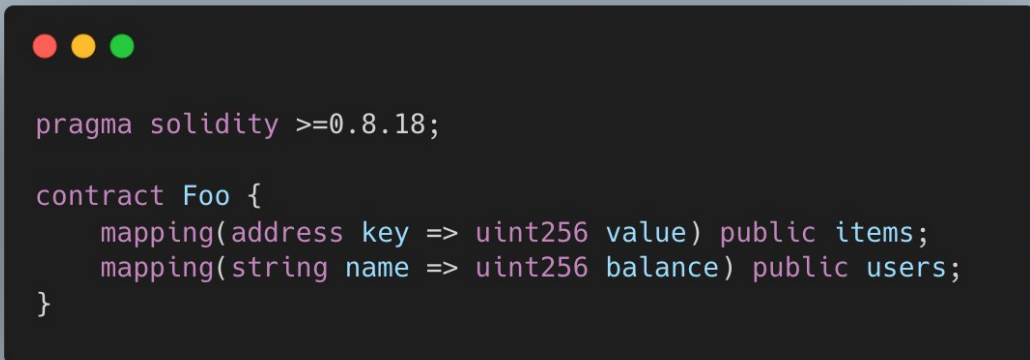
## 6. Use named mappings

Severity: **Informational**

Files: \*.sol

### Description

Solidity 0.8.18 and later support a language feature called [named mappings](#), where keys and values can be named to give devs better documentation of what's expected in each.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains Solidity code that demonstrates the use of named mappings. The code is as follows:

```
pragma solidity >=0.8.18;

contract Foo {
    mapping(address key => uint256 value) public items;
    mapping(string name => uint256 balance) public users;
}
```

### Impact

Extraneous comments that could be inlined.

### Recommendation

Experiment with named mappings where the storage layout otherwise requires comment explanations.

### Response

Not implemented.

## 7. Use block.chainid instead of embedded assembly

Severity: **Informational**

Files: PaintingVoucher.sol, RuffleVoucher.sol

### Description

Latest versions of Solidity expose `block.chainid` as a globally available block property without needing to use a Yul helper function.

### Impact

Code bloat.

### Recommendation

Delete the `getChainID()` functions, and replace all calls to them with a simple inline `block.chainid` variable reference.

### Response

Not implemented.

## 8. Use custom errors instead of require strings

Severity: **Informational**

Files: \*.sol

### Description

Custom errors are slimmer from a deployment codesize perspective, and also use less gas on reverts. They're also better for grouping similar errors into canonical categories without the repetition of similar-but-distinct revert strings. Compiler versions will need to be ^0.8.4 or higher instead of ^0.8.0 for compatibility here.

### Impact

Lower deployment and usage gas costs.

### Recommendation

Update require statements to revert with custom errors.

### Response

Not implemented.

## 9. Remove pragma abicoder v2

Severity: **Informational**

Files: PaintingVoucher.sol, RuffleVoucher.sol

### Description

The abicoder v2 is enabled by default in Solidity 0.8 and later, see <https://docs.soliditylang.org/en/latest/layout-of-source-files.html#abi-coder-pragma>.

### Impact

Unnecessary default declarations.

### Recommendation

Remove.

### Response

Not implemented.

## 10. Fix typos

Severity: **Informational**

Files: \*.sol

### Description

Replace “RuffleVoucher” with “RaffleVoucher”

Replace “tradable” with “tradeable”

“whiteList” and “waitList” are each one word, can be replaced with “whitelist” and “waitlist”

### Impact

Cleaner code.

### Recommendation

Update spelling.

### Response

Fixed.



## 11. Standardize msg.sender and \_msgSender() usage

Severity: **Informational**

Files: \*.sol

### Description

Both the global variable msg.sender and the OpenZeppelin context helper \_msgSender() are used interspersed throughout the code. This will lead to incompatibility if use of \_msgSender() is desired, and leads to extra gas consumption if use of msg.sender is desired.

### Impact

Incompatibility and unnecessary gas usage.

### Recommendation

Choose one format and stick with it.

### Response

\_msgSender() used everywhere now.

## 12. Standardize license usage

Severity: **Informational**

Files: \*.sol

### Description

Both the MIT License and the Unlicense are interspersed throughout the code.

### Impact

Unclear licensing.

### Recommendation

Choose one license and remain consistent throughout the repo.

### Response

Fixed.

## 13. Replace private mappings + public getters with public mappings

Severity: **Informational**

Files: Painting.sol, Android.sol

### Description

Solidity automatically populates getter functions for public mappings. So it's cleaner to replace private mappings that expose one-to-one getter functions with a public mapping. In Painting.sol this can be done for `\_parents`, `\_claimers`, `\_ipfsLinks`, and `\_collections`. In Android.sol, this can be done for `\_genes`.

### Impact

Code bloat.

### Recommendation

Simplify these into public mappings.

### Response

Not implemented.