NetraNFT Smart Contract Security Audit

v2.0.0

Table of Content

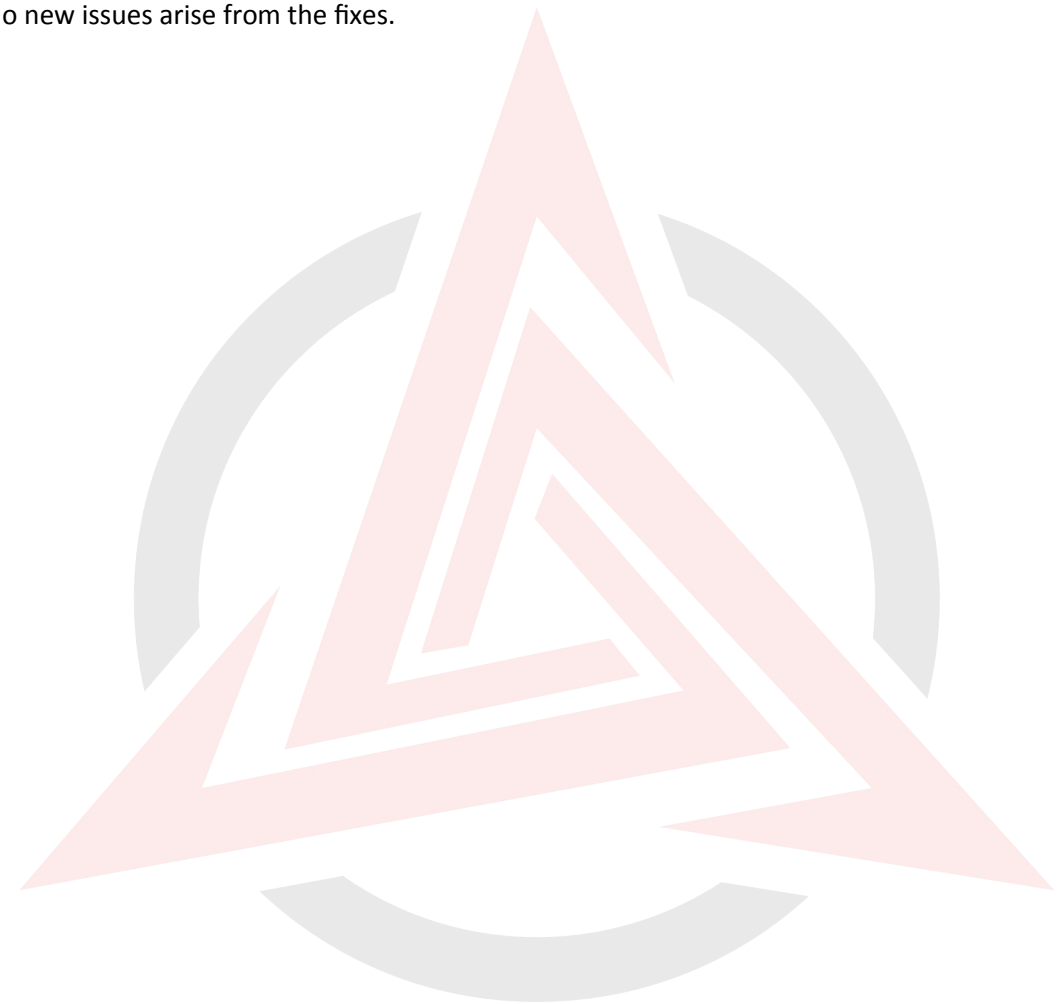
Disclaimer

This audit report does not provide any warranties or guarantees on the vulnerability-free nature of the given target(s), nor do they provide any indication of legal compliance. The conducted security audit process aims to reduce security risks possibly implemented in the smart contracts. This audit report aims to improve the code security and quality of the given target(s) and is not able to detect any security issues that will occur in the future. This audit report should not be considered as financial investment advice.

Assessment Target

The security analysis targets the NetraNFT Solidity smart contract available on Github at commit a1c2df9. The project consists of one main contract in NetraNFT.sol and several supporting contracts and libraries. The code is written in Solidity.

The target was fixed and re-examined at commit 4991d65 to ensure all issues have been addressed and no new issues arise from the fixes.



Project General Overview

Netra is a platform that utilises NFT for sharing music ownership between artists and investors. Using NetraNFT, one musical work's rights (such as future income from streaming platforms of a music recording) is split into tiers, where in each tier there will be tokens with unique IDs. One tier represents a collection of tokens with identical rights and perks.

NetraNFT allows limited early minting by whitelisted addresses, with a maximum of two tokens each account. Once a specified time passes, the minting function is enabled to everyone.

Security Audit Specification

Approach and Methodologies

The security audit was performed according to the known security best practice which may include the following techniques:

1. Static Analyses

   Static analyses are performed using automated tools to identify known security issues and to perform a quick survey on the target(s). The tools used include:

   - Mythril
   - Slither
   - Surya

2. Dynamic Analysis

   Dynamic analysis is performed using a white-box method where the source codes are known and manually analyzed for any security issues.

3. Tests

   Security-focused tests are written to ensure that the target(s) work as intended. Edge-cases, duplications, mass-usage, and proof-of-concepts of certain hypotheses are implemented into the tests for completeness and coverage.

4. Fuzzing

   Fuzzing is used to identify certain inputs that may break the target(s) component(s) due to inaccurate security assumptions or a lack of input sanity checks.

Goals

The focus of the audit is to verify the security, resiliency, and correctness of the target(s) based on the developer's specifications.

Security

Each issue is categorized into a severity level based on OWASP's Risk Rating Methodology which focuses on Impact and Likelihood of the risk.

- Critical
- High
- Medium
- Low
- Note

The relationship between Impact and Likelihood is shown in the Appendix.

Findings Summary

The security analysis discovers a total of 4 issues: one issue categorized as critical, two medium, and one note, as shown in the table below.

Critical	High	Medium	Low	Note
1	0	2	0	1

Below is the summary and status of all issues.

Id	Category	Status
1	Critical	Fixed
2	Medium	Fixed
3	Medium	Closed with Improvement
4	Note	Fixed

The detail of the issues will be presented in the following sections.

1. Incorrect Token ID Generation

Related Asset(s)	NetraNFT.sol
Category	Critical
Impact	High
Likelihood	High
Status	Fixed

Description

Function *`generateId()`* assists in generating unique IDs for minting new tokens (through function *`mint()`*). The current implementation generates a new number by adding the previous tier's supply with the current supply. However, this method is only valid when there are only two tiers. A contract with more than two tiers will not be able to generate tokens in tier higher than two, because the function *`mint()`* reverts when a newly generated ID exists.

Since the project intends to have at least three tiers (based on the current documentation), this bug will have a great impact to the token minting ability. Namely, the contract cannot mint any tokens beyond tier two.

Recommendation

We recommend refactoring the current implementation by considering the number of tiers and how many tokens will be generated in each tier. Alternatively, the contract can store initial IDs for each tier during initialization phase, assuming there will not be any additional tiers added afterwards.

Fixes

The issue was fixed in PR#1 by recording *`startId`* on each tier to ensure correct IDs are generated.

2. Off-by-One Error

Related Asset(s)	NetraNFT.sol
Category	Medium
Impact	Low
Likelihood	High
Status	Fixed

Description

Function *validateWhitelist()* is used to validate the purchase during an early minting by whitelisted addresses. It is called by function *mint()*. One of the checks in function *validateWhitelist()* is ensuring that the amount sold is not greater than the allocated amount for sale by whitelisted address. However, there is an off-by-one error caused by L369 due to an incorrect comparison operator. The bug causes the purchase by whitelisted addresses to exceed the allocated amount by one.

Recommendation

We recommend changing the comparison operator in L369 from “>” to “>=” such that the line would appear as:

```
if (_tier.amountSold >= _tier.whitelistAllocation) {
```

Fixes

The issue was fixed in PR#2. The recommended change was implemented.

3. Potential Revert on Ether Transfer to Contract

Related Asset(s)	NetraNFT.sol
Category	Medium
Impact	Medium
Likelihood	Low
Status	Closed with Improvement

Description

Function *withdrawEther()* allows sales distribution to the beneficiaries, namely the artist and Netra. The artist will receive according to a a percentage defined during contract construction (*artistShare*). While Netra will receive the remaining amount.

This function uses a low level *call* to transfer funds in Ether to the artist and Netra. If one of the beneficiaries is a contract and the contract is unable to receive the funds, then this function will always revert (due to check on L170 and 173) and the funds will be unrecoverable.

Recommendation

We also recommend ensuring that both artist and Netra addresses are not contracts to make sure that each low level *call* does not revert. We also recommend adding reentrancy guard on function *withdrawEther()* to reduce security risk from reentrancy.

Fixes

Reentrancy protection was added in PR#3.

4. Others

Related Asset(s)	NetraNFT.sol
Category	Note
Impact	Low
Likelihood	Low
Status	Fixed

Description

The following findings do not directly impact the security of the system but might be useful.

1. **Typo**

   L32: RoyaltyWithrawn -> RoyaltyWithdrawn
   L47: InsuficientRoyalty -> InsufficientRoyalty
   L53: InsuficientPayment -> InsufficientPayment

Recommendation

We recommend taking the suggested steps to address the finding(s) and improve the system.

Fixes

The issues were fixed in PR#4.

Appendix: OWASP Severity Rating

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			