Art Portfolio | Zyphernix Security Review

Smart Contract Audit Report

Prepared by: Zyphernix Security

DATE OF ENGAEMENT: 7th May2025 - 18th May 2025

Introduction

During the period of **May 7, 2025 to May 18th, 2025** – Zyphernix Security Team performed a security audit for **Art Portfolio** smart contract.

The code for the audit was taken from following the official link**:**

https://github.com/jan-miksik/art-portfolio-contracts/blob/main/src

[Commit hash: b61869d4627a4379d5e8b0e785c0b3ed3c3dea05]

Scope of Audit

The scope of this audit was to analyze and document the Art Portfolio smart contract codebase for quality, security, and correctness. Following is the list of smart contracts included in the scope of this audit:

- ✓ VariousPicturesCollection.sol

OUT-OF-SCOPE: External contracts, External Oracles, other smart contracts in the repository or imported smart contracts.

About Art Portfolio

The contract is meant for personal art creation and publishing NFTs, because of that some parts of the contract may give a centralized impression. The contract should allow gradual minting of NFTs.
The general idea is once the image for the NFT is created, the owner mints the NFT and then lists it on marketplace, adds to auction or anything else.
The contract doesn't limit NFT amounts. The collection is meant for releasing NFTs gradually over time without any specific vision about the final amount.

Roles

- **Owner**: Author of the NFTs or maintainer of the contract. Can mint NFTs. Has responsibility for collection metadata, can eventually update them if necessary. Can renounce or move ownership of the contract to somebody else. Can set up Royalty receiver. Can change royalties percent.
- **Collector**: Owns NFTs of the Various Pictures Collection. Buys these NFTs primarily through marketplaces. Can burn NFTs. Should pay royalties if selling NFTs, however this is not enforced and if collector and the marketplace allow it, it is possible to sell without royalties.
- **Outsider**: Does not own NFTs of the Various Pictures Collection. Should not be able to do any writing operation on the contract.

Techniques and Methods

The following techniques, methods, and tools were used to review all the smart contracts.

- Research into architecture and purpose
- Smart Contract manual code read and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual Assessment of use and safety for the critical solidity variables and functions in scope to identify any arithmetic-related vulnerability classes.
- Scanning of solidity files for vulnerabilities, security hotspots, or bugs. (MythX)
- Static Analysis of security for the scoped contract and
- imported functions. (Slither)
- Smart Contract analysis and automatic exploitation (teEther)
- Symbolic Execution / EVM bytecode security assessment (Manticore)

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of token standards.
- Efficient use of gas.
- Code is safe from reentrancy and other vulnerabilities.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Surya, Solhint, Mythril, Slither, Solidity static analysis, Foundry.

Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High -** leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium -** only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low -** can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.
- **Informational** – No impact - coding practices, design choices, or implementation details.

Likelihood

- **High -** attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- **Medium -** only a conditionally incentivized attack vector, but still relatively likely.
- **Low -** has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Action required for severity levels

- **Critical -** Must fix as soon as possible (if already deployed)
- **High -** Must fix (before deployment if not already deployed)
- **Medium -** Should fix
- **Low -** Could fix
- **Informational** – Could fix

Assessment Summary and Findings Overview

Every issue in this report has been assigned with a severity level. There are four levels of severity – **CRITICAL**, **HIGH, MEDIUM, LOW and INFORMATIONAL**.

Summary of issues found

TYPE	REPORTED
CRITICAL	0
HIGH	0
MEDIUM	0
LOW	0
INFO	5
TOTAL	5

ID	TITLE	SEVERITY	STATUS
I-01	No check whether tokenURImetadata is already exists	INFO	**Acknowledged**
I-02	Metadata can be changed post-minting	INFO	**Acknowledged**
I-03	Use immutable variables for constants like MAX_ROYALTY_BASIS_POINTS	INFO	**Acknowledged**
I-04	URIs are not checked for validity, risking broken links or incompatible formats	INFO	**Acknowledged**
I-05	Add a minimum royalty requirement	INFO	**Acknowledged As per design.**

Findings

Critical Findings
None

High Findings
None

Medium Findings
None

Low Findings
1. **[I-01] No check whether tokenURImetadata is already exists**
   The contract allows the owner to mint multiple NFTs with the same tokenURImetadata URI. Since the URI points to the NFT's metadata (e.g., JSON with name, description, image), this could result in multiple NFTs sharing identical metadata, effectively creating duplicates within the collection.

   Recommendation: To prevent duplicate tokenURImetadata, you can add a check to ensure that each tokenURImetadata is unique before assigning it to a new token. This involves tracking used URIs and verifying that a new URI hasn't been used previously.

   Developer's Response: Even though it's unlikely in this type of collection, it could still be desired to allow duplicates within the collection. The NFTs will then have different token IDs. In case of a mistake, the metadata can be edited by the owner.

2. **[I-02] Metadata can be changed post-minting**
   The contract allows the owner to change metadata post-minting. Make token URIs immutable by default unless explicitly allowed by the owner during minting. This could involve a flag in the minting functions to indicate whether the URI can be updated.

   Developer's Response: It's an understandable concern. However, the protection against mistakes and broken links in the future outweighs the benefits of immutability. For reference, major NFT projects also allow post-minting changes. The function for changing metadata emits an event, which allows easier tracking of these changes.

3. **[I-03] Use immutable variables for constants like MAX_ROYALTY_BASIS_POINTS**
   Using immutable variables for constants like MAX_ROYALTY_BASIS_POINTS optimizes gas usage and enhances security by ensuring the value cannot be modified.

Developer's Response: Constant should be more gas efficient than immutable. Since the value is inlined at compile time, it consumes zero runtime gas when accessed.

4. **[I-04] URIs are not checked for validity, risking broken links or incompatible formats**
   Add validation to ensure URIs point to valid JSON or resolve to accessible content. For example, check that image URIs have valid prefixes (ipfs://, https://) or use an off-chain oracle to verify content.

   Developer's Response: It's hard to predict all possible formats of URIs in the future. The safeMintWithURI function is intended for more expert use. Validation could be limiting here.

5. **[I-05] Add a minimum royalty requirement**
   Adding a minimum royalty requirement means enforcing a lower bound on _royaltyBasisPoints to ensure the royalty percentage is at least a specified value, such as 1% (100 basis points). This check would apply when setting the initial royalty in the constructor and when updating it via updateRoyaltyBasisPoints.

   Developer's Response: As per design. Royalty can be zero.

Closing Summary

Overall, the smart contract code is well documented.

Several issues of were found and reported during the audit. The outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verification of essential properties were performed to achieve objectives and deliverables set in the scope. Zyphernix Security recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts.

Disclaimer

Zyphernix Security audit is not a security warranty, investment advice, or an endorsement of the **Art Portfolio smart contracts**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **Art Portfolio Team** put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.