min3.xyz

Smart Contracts Audit

Oleksandr Zadorozhnyi, Viacheslav Vozniuk

June, 2023

Email: info@4irelabs.com

Website: 4irelabs.com

CONFIDENTIAL

This document and the Code Review it references is strictly private, confidential and personal to its recipients and should not be disclosed, copied, distributed or reproduced in whole or in part, not passed to any third party.

1. **INTRODUCTION.** min3.xyz requested 4irelabs a review of the contract implementing their ERC721 token.

   There is 1 contract source code in scope:
   https://sepolia.etherscan.io/address/0xa32933576371988bc1c462748e4ee9f1a6714893#code

   Updated contract:
   https://sepolia.etherscan.io/address/0x4b8daad4a19b8d786c612dcbda2f119ab171776b#code

2. **WARRANTY.** Audit is provided on an "as is" basis, without warranty of any kind, express or implied. The Auditor does not guarantee that the Code Review will identify all instances of security vulnerabilities or other related issues.

3. **EXECUTIVE SUMMARY.** Team resolved or acknowledged all founded issues. There are no known compiler bugs, for the specified compiler version, that might affect the contracts logic.

4. **CRITICAL BUGS AND VULNERABILITIES.** No Critical and Major issues have been found.

5. LINE BY LINE REVIEW.

5.1. OgPremiumNft.sol:

5.1.1. **Resolved** Line 159. Medium: *AIRDROP_BASEURI_MANAGE_ROLE* could mint more than 40 NFTs, while documentation states contrary. *airdrop()* function checks only that *totalSupply* would not be greater than 5000 after airdrop minting, this allows *AIRDROP_BASEURI_MANAGE_ROLE* arbitrary number of NFTs between 1 and 5000 during arbitrary number of calls. Also worth mentioning that *airdrop()* minting could be performed in any stage of sale - *airdrop()* function does not check current activeTier value.

Recommendation: Consider adding checks to airdrop() function that would not allow AIRDROP_BASEURI_MANAGE_ROLE to mint more than 40 NFTs.

Update: Team recognizes AIRDROP_BASEURI_MANAGE_ROLE's possibility to mint more than 40 NFTs in arbitrary time as intended behavior.

5.1.2. **Resolved.** Line 159 Medium: *AIRDROP_BASEURI_MANAGE_ROLE* could bypass *maxSupply* check in *airdrop()* function using reentrancy in *_safeMint()*. Contract would call *recipients[i]* address in case it's a smart contract *using _checkContractOnERC721Received* callback function. In this case possible reentrancy call to airdrop() and minting additional NFTs while *totalSupply()* is still less than 5000. After finishing all mint calls *totalSupply()* could become greater than 5000, which would break contract requirements.

Recommendation: Consider adding nonReentrant modifier to airdrop() function.

5.1.3. Resolved. Line 179. Medium: Incorrect implementation of *supportsInterface()* function for ERC721A contract. Current implementation would return false value for example to calls like *supportsInterface(IERC721interfaceId)*. While the contract still is a valid ERC721, there could be problems during integrating this project with some external protocols that require the contract to return true value on appropriate *supportsInterface()* calls. Additional info about this problem:

https://chiru-labs.github.io/ERC721A/#/migration?id=supportsinterface

Recommendation: Consider updating supportsInterface() function in the way the link above suggests.

5.1.4. Resolved. Line 89. Low: *getLatestPrice()* does not implement sanity checks on Chainlink priceFeed return data. This could lead to using stale/incorrect prices. Consider adding sanity checks which at least checks that price > 0, roundId and timestamp != 0. Additional info:

https://docs.chain.link/data-feeds#check-the-timestamp-of-the-latest-answer

5.1.5. Resolved. Line 147, 159. Low: Inconsistency in emitting *MintEvent*, while *allowlistMint()* function emits such an event, *mint()* and *airdrop()* functions miss it.

5.1.6. Acknowledged. Note: NatSpec comments are missing in most parts of the code.

5.1.7. Resolved. Line 2. Note: Use locked compiler version to prevent unintended deployment with outdated/too fresh compiler version.

5.1.8. **Resolved.** Line 14. Note: Using of SafeMath is not needed since Solidity version 0.8.0 since the compiler checks overflow/underflow cases.

5.1.9. **Resolved.** Line 16, 34, 35. Note: variables *priceFeed, maxSupply, royaltyFeeNumerator* could be immutables/constants since they are never changed after deployment.

5.1.10. **Resolved.** Line 46. Note: Wrong hardcoded Chainlink priceFeed address. Contract deployed to Sepolia testnet, while uses Chainlink priceFeed from Goerli testnet. This address must be updated to appropriate before Mainnet deployment.

5.1.11. **Resolved.** Line 42, 46, 50, 53. Note: Consider using constant variables for hardcoded addresses instead of using "magic" values.

5.1.12. **Resolved.** Line 74, 78, 81, 85. Note: Missing emitting events on sensitive changes.

5.1.13. **Acknowledged.** Line 101. Note: Since *verifyPayment()* allows payment amount to be greater than NFTs cost, this could lead to some users overpaying in case of wrong input on rapid price changes. Consider updating mint logic in a way that excess value returns to users.

5.1.14. **Resolved.** Line 152. Note: Open *TODO* comment.

5.2. Documentation mismatches:

5.2.1. In "Randomisation" section encountered more than 5000 NFTs (5 x Gold + 1,040 x Silver + 3,995 Bronze = 5040).

5.2.2. While documentation expects that each tier stage would last 1 week and would be moving from 0 -> 1 -> 2 -> 3 -> 0, it's not restricted on contract level. Contract owner could arbitrarily switch the current tier anytime. Ignore this in case it's intended behavior.

5.2.3. Described airdrop flow suggests to switch tier value to "3" (public mint) before calling airdrop() function. This action is not needed since the airdrop() function does not check the current tier. Also, it's dangerous since some actors could monitor transactions on contract and call the mint() function before the intended stage.

5.2.4. "Traits" section uses the CryptoPunks collection as a reference. Worth mentioning that this collection is not implementing ERC721 standard (since it's older than standard) and it could be dangerous to use it as reference since most marketplaces and other protocols treat this collection (and some other old ones) in exceptional ways.

6. MIN3.XYZ SMART CONTRACTS AUDIT UPDATE:

6.1. Contracts have been updated and issues (5.1.1, 5.1.2, 5.1.3, 5.1.4, 5.1.5, 5.1.7, 5.1.8, 5.1.9, 5.1.10, 5.1.11, 5.1.12, 5.1.14) have been resolved.

6.2. All documentation mismatches acknowledged or recognized as intended.