Smart Contract Security Audit:

Next Earth

Author: *six from Awalcon*
Date: *2021.07.07.*

Table of Contents

Disclaimer

The list of findings and recommendations are summarized in the Audit Results.

The matters raised in this report are only those identified during the review and are not necessarily a comprehensive statement of all weaknesses that exist or all actions that might be taken. This work was performed under limitations of time and scope that are not potentially relevant to the actions of a malicious attack.

The review is based at a specific point in time, in an environment where both the systems and the threat profiles are dynamically evolving. It is therefore possible that vulnerabilities exist or will arise that were not identified during the review and there may or will have been events, developments and changes in circumstances subsequent to its issue.

The security analysis is purely based on the provided smart contracts alone. No other products or systems have been reviewed. The purpose of the audit is to identify issues related to the logic and quality of the code.

Overview

The Next Earth project requested a smart contract code security audit.

Start date of the audit: 2021.07.06.

Report date: 2021.07.07.

Project website: https://nextearth.io/
Platform: Solidity / Ethereum

Audited commit: *4bac0998cac7e19a3a5370c997551ba71bb82d57*
Smart contracts in scope:

- NFT.sol
- Payment.sol
- Presalse.sol
- PriceFeed.sol

Imported smart contracts:

- @chainlink/contracts/src/v0.6/interfaces/AggregatorV3Interface.sol
- @openzeppelin/contracts/access/AccessControl.sol
- @openzeppelin/contracts/access/Ownable.sol
- @openzeppelin/contracts/security/Pausable.sol
- @openzeppelin/contracts/token/ERC721/ERC721.sol
- @openzeppelin/contracts/token/ERC721/extensions/ERC721Burnable.sol
- @openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol
- @openzeppelin/contracts/utils/cryptography/ECDSA.sol

Overall result: pass

Auditor: six ~ *PGP 450F 4AC8 0BD8*

Objective and methodology

The objective of the security assessment is to gain insight into the security of the smart contracts listed in the scope.

Code review main check items:

- Line-by-line audit
- Business logic
- Data consistency
- Coding style violations
- Gas usage
- Reentrancy

Automated tools:

- Static analysis with Slither
- Symbolic execution with Manticore and Mythril

Further documents incorporated in the methodology:

- Smart Contract Weakness Classification Registry - https://swcregistry.io/
- https://github.com/miguelmota/solidity-audit-checklist

Audit results

Critical severity

No cricitcal severity issue have been found during the manual code review or by using automated tools.

High severity

No high severity issue have been found during the manual code review or by using automated tools.

Medium severity

No medium severity issue have been found during the manual code review or by using automated tools.

Low severity

Presale.sol | Creation of unintended packs in setPackPrice()

Description and impact

At line61 it is specified that we have packages from 1 to 5, but in the setPackPrice function at line120, it allows to pass zero or any positive number up to *2**32-1 for _type* and *2**256-1 for price*.

Logic says, that would be also an integer overflow, but from Solidity compiler version 0.8.0, overflowing transactions get reverted automatically.

This vulnerability allows the creation of packages against the intentional logic of the project.

This is only a low level vulnerability as onlyOwner is used and it can't be exploited by others.

Line 61.:

```
require(packType >= 1 && packType <= 5, "invalid pack type"); // we have 5 pack types from 1 to 5
```

Line 120.:

```
function setPackPrice(uint32 _type, uint256 price) external onlyOwner {
    packPrices[_type] = price;
}
```

Proposed solution

You could use a require statement checking the type (and preferably the price too) before setting the price for a pack.

Presale.sol | Unused code, todo, typo

Description

Payment.sol, line36.

// TODO can we do this pull over push?

Presale.sol, line 88.

// uint256 contractShare = msg.value - charityShare - comissionShare; // not used, ***need to remove it***

NFT.sol, line 42.

// happens against a ~~***signle***~~ *single user...*

Presale.sol, multiple lines:

“comissionCode” → “commissionCode”

Proposed solution

Update the mentioned points.

NFT.sol | Minting functions can go above gas limit

Description

The issue was acknowledged by the project before the audit.

Function safeMint from line 35. and function safeMintTo() from line 49. can go above the gas limit if *tokenIds.length* becomes too big.

Proposed solution

Don’t let users break themselves.

Lack of README file

Description

The project does not have a README file or documentation.

Proposed solution

Provide a clear documentation and use more comments.

Lack of comments regarding functionality

Description

The smart contracts have comments at some critical points, but not about the functions or general code logic.

It is also recommended to add NatSpec to the code.

Proposed solution

Follow the Solidity Coding style guide.

https://docs.soliditylang.org/en/latest/style-guide.html

Smart contract dissection - functions

NFT.sol: function pause()

// Passed

NFT.sol: function unpause()

// Passed

NFT.sol:function safeMint(uint256[] calldata tokenIds, bytes calldata sig)

// Passed after intended logic double checked with Silur

NFT.sol: function safeMintTo(address to, uint256[] calldata tokenIds)

// Passed after intended logic double checked with Silur

NFT.sol: function firstOwnerOf(uint256 tokenId)

// Passed

NFT.sol: function baseURI()

// Passed

NFT.sol: function setBaseURI(string calldata _baseUri)

// Passed

NFT.sol: function _beforeTokenTransfer(address from, address to, uint256 tokenId)

// Passed

NFT.sol: function supportsInterface(bytes4 interfaceId)

// Passed

Payment.sol: function buyToken()

// Passed after intended logic double checked with Silur

Payment.sol: function getBeneficiaries()

// Passed

Payment.sol: function pause() public

// Passed

Payment.sol: function unpause() public

// Passed

Payment.sol: function setupOwners()

// Passed

Payment.sol: function setupRatios()

// Passed

Payment.sol: function crytic_beneficiary_balance_not_decreasing()

// Passed

Payment.sol: function crytic_buyer_balance_not_decreasing()

// Passed

Presalse.sol: function buyPack(uint32 packType, bytes32 landsRoot, uint16 comissionPercent, string calldata comissionCode, uint16 discountPercent, bytes calldata sig)

// Passed

Presalse.sol: function getPurchases()

// Passed

Presalse.sol: function getBalance()

// Passed

Presalse.sol: function setDiscount(address _addr, bool _isDiscounted)

// Passed

Presalse.sol: function isDiscounted()

// Passed

Presalse.sol: function pause()

// Passed

Presalse.sol: function unpause()

// Passed

Presalse.sol: function setPackPrice(uint32 _type, uint256 price)

// Passed with low

Presalse.sol: function claimCharity()

// Passed

Presalse.sol: function claimComission(string calldata code, uint256 amount, bytes calldata sig)

// Passed

Presalse.sol: function claimSale()

// Passed

Presalse.sol: function setCharityAddress(address payable _address)

// Passed

Presalse.sol: function getBNBPrice(uint256 dollarAmount)

// Passed

PriceFeed.sol: function decimals()

// Passed

PriceFeed.sol: function description()

// Passed

PriceFeed.sol: function getRoundData(uint80 _roundId)

// Passed

PriceFeed.sol: function latestRoundData()

// Passed

PriceFeed.sol: function version()

// Passed

Contact

Awalcon - six

Website: *https://awalcon.org/*

E-mail: *six@awalcon.org*

Telegram/Signal: *+36 20 256 4090*

Git: *https://git.hsbp.org/six*

PGP*: B1F7 B1D6 8838 98B4 2212 1D90 CA71 D1E4 078E 99C5*