Guard

Smart contracts security assessment

Final report

Tariff: Standard

Naughty Giraffes

March 2022

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for Naughty Giraffes.

Standard ERC721 token contract with an added author tip feature. ERC721 interface is realized with the use of OpenZeppelin libraries, which is considered the best practice.

The contract allows buying NFT (mints tokens with unique id) on presale, early sale and public sale for different prices. There are minting limits per buyer address and supply limits on each sale step. The contract owner has ability to set price, supply, mint limits and time of sales at any time.

Owner must reveal token URI at some point in time.

The md5 sum of the file with contract code is a2bc0e0ba65c6cb228860cc9feff2bd0.

Name	Naughty Giraffes
Audit date	2022-03-16 - 2022-03-16
Language	Solidity
Platform	Ethereum

Contracts checked

Name	Address
NaughtyGiraffes	

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyse smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed
Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed
Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed

Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	not passed
Outdated Compiler Version	passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity	High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.
Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. Withdrawal address is not set (NaughtyGiraffes)

The state variable `withdrawalWalletAddress` is not defined. Thus, all earned ethers can be lost when calling function `withdraw()`.

Recommendation: Set the address in the contract constructor.

Medium severity issues

No issues were found

Low severity issues

1. Floating pragma (NaughtyGiraffes)

Pragma should be fixed to the version that the contracts are expected to be deployed with. This helps to avoid deploying using an outdated compiler version and shields from possible bugs in future solidity releases.

2. Gas optimization (NaughtyGiraffes)

a. The functions `setMerkleroot()`, `setStartTime()`, `setMaxMintPerAddress()`, `setMintPrice()`, `setMaxSupply()`, `withdraw()`, `tokensInWallet()`, `tokenURI()` can be declared as `external` to save gas.

b. The argument of the functions `setMintPrice()`, `setMaxSupply()` can be declared as `calldata` to save gas.

3. Validation of input parameters (NaughtyGiraffes)

We recommend making validation of input parameters for the functions `setStartTime()`, `setMaxMintPerAddress()`, `setMintPrice()`, `setMaxSupply()` to prevent entering incorrect

values.

4. Redundant modifier (NaughtyGiraffes)

The modifier `isLive()` duplicates conditions of the `isAddressOnWhitelist()` modifier and can be deleted.

5. Unused argument (NaughtyGiraffes)

The argument `amount` of the function `withdraw()` is not used. At the same time, with each function call, the entire balance is transferred.

```
    function withdraw(uint256 amount) public onlyOwner {
        require(address(this).balance >= amount,"Add correct eth amount to withdraw.");
        uint256 balance = address(this).balance;
        payable(withdrawalWalletAddress).transfer(balance);
    }
```

Conclusion

Naughty Giraffes NaughtyGiraffes contract was audited. 1 high, 5 low severity issues were found.

We strongly recommend writing unit tests to have extensive coverage of the codebase minimize the possibility of bugs and ensure that everything works as expected.

The contracts are quite strongly dependent on the owner's account. Users interacting with the contracts must trust the owner.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability)set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

0x Guard