verichains

SECURITY AUDIT OF

OP3N SMART CONTRACTS

Public Report

Jan 19, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Jan 19, 2022. We would like to thank the OP3N for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the OP3N Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About OP3N Smart Contracts

OP3N is THE platform for creators & fans. With OP3N, musicians, filmmakers, gamers, and artists can host content and metaverse experiences through NFT Memberships.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the OP3N Smart Contracts.

It was conducted on commit b41d45d0123b099efe67ebbf4cd5340f20943a36 from git repository *https://github.com/EST-Media/op3n-nft-audit-contracts*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
3eb993e86edb85fe6635ef0bd7335f3f54775c557f820869c009e6eafca25483	**INFTPackage.sol**
a05d41c877f46f9e90fc710e9c9717f649d45410f8ec34382aab25c044d36377	**INFTPackageTrade.sol**
df686efb648c2080ec4c16f6fd08cb9df028e1cbb9532845b00634093f9e68da	**INFTUpgradeable.sol**
f2e2a9d07f45c3fecfd0a4183d8f3e7cfd026ce260a0e46391136e64912009e4	**NFTFactory.sol**
456268ddae32d7f8a78f7a0024e6f2d8fa1c15e62feed62af7e04dad4abac5ba	**NFTPackage.sol**
f257134140254509bc55e0034e0943ba69b4b786f0f4766023e79698a6772661	**NFTPackageTrade.sol**
4c0468ca36b214fff063ce0abf6f7eeb33c6fd61f48e423688e35cf9dfd050f3	**NFTUpgradeable.sol**

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow

- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

OP3N Smart Contracts contains 4 main contracts: NFTUpgradeable, NFTPackage, NFTPackageTrade and NFTFactory.

2.1.1. NFTUpgradeable contract

This is the NFT upgradable contract in the OP3N Smart Contracts, which extends ERC721Upgradeable contract. The contract implements RBAC (Role Based Access Control) to restricting system access to authorized users. By default, the owner can set any role for anyone. Addresses with PAUSER_ROLE can pause/unpause contract, users can only transfer NFT when the contract is not paused.

2.1.2. NFTPackage contract

This contract extends NFTUpgradeable contract and is used for manage packages. Each package has a limited number of NFTs configured by the owner. Addresses with MINTER_ROLE can mint new NFT for the package.

2.1.3. NFTPackageTrade contract

This is an upgradable contract where users can buy NFTs. Orders are generated and signed in backend server. When users pay tokens or native token to process the payment, the contract will validate the order to check for a valid signature, take fee, transfer payment to order's recipient and mint NFT to the buyer using NFTPackage contract.

2.1.4. NFTFactory contract

This contract implements beacon proxy which allows multiple proxies to be upgraded to a different implementation in a single transaction. It will help create upgradable contracts in OP3N Smart Contracts.

2.2. Findings

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

OP3N fixed the code according to Verichains's draft report in commit eff388ac21601fc8048514ca780789ddae4705ba.

2.2.1. NFTPackageTrade.sol - Fee is not collected CRITICAL

In mint function, when token is used to pay, only transferAmount (order.amount - order.fee) is sent to creator, order.fee is not collected. In addition, when paying with native token, order.fee is sent to creator instead of _feeReceiver.

```
address private _feeReceiver;

function mint(Order memory order, bytes memory sig) public payable overri…
  de {
    address _verifier = recoverVerifier(order, sig);
    require(verifiers[_verifier], "403");

    address payable creator = payable(INFTUpgradeable(order.project).crea…
  tor());
    uint256 transferAmount = order.amount - order.fee;
    if (address(0) == order.token) {
        require(order.amount <= msg.value);
        Address.sendValue(creator, transferAmount);
        if (0 < order.fee) {
            Address.sendValue(creator, order.fee);
        }
    }   else {
        _transferToken(order.token, msg.sender, creator, transferAmount);
        if (0 < msg.value) {
            Address.sendValue(creator, msg.value);
        }
    }

    INFTPackage(order.project).safeMintTo(order.to, order.package, order.…
  uri);
}
```

RECOMMENDATION

Sending fee to feeReceiver.

UPDATES

- *Jan 18, 2022*: This issue has been acknowledged and fixed by the OP3N team.

2.2.2. NFTPackageTrade.sol - Signature replay HIGH

In the mint function, signature is not checked whether it is used or not. User can replay this signature and mint many times.

```
function mint(Order memory order, bytes memory sig) public payable overri…
  de {
    address _verifier = recoverVerifier(order, sig);
    require(verifiers[_verifier], "403");

    address payable creator = payable(INFTUpgradeable(order.project).crea…
  tor());
    uint256 transferAmount = order.amount - order.fee;
    if (address(0) == order.token) {
        require(order.amount <= msg.value);
        Address.sendValue(creator, transferAmount);
        if (0 < order.fee) {
            Address.sendValue(creator, order.fee);
        }
    }   else {
        _transferToken(order.token, msg.sender, creator, transferAmount);
        if (0 < msg.value) {
            Address.sendValue(creator, msg.value);
        }
    }

    INFTPackage(order.project).safeMintTo(order.to, order.package, order.…
  uri);
}
```

RECOMMENDATION

The order signature should be checked to ensure that it is used once.

UPDATES

- *Jan 18, 2022*: This issue has been acknowledged and fixed by the OP3N team.

2.2.3. NFTUpgradeable.sol, NFTPackage.sol - Unused modifier onlyOwner INFORMATIVE

There is onlyOwner modifier in the contract but the contract check for owner() == msg.sender everywhere instead of using the modifier.

```
// NFTUpgradeable.sol
modifier onlyOwner() {
    require(owner() == msg.sender);
    _;
}

function transferOwnership(address newOwner) public virtual override {
    require(owner() == msg.sender);
    require(newOwner != address(0));

    _owner = newOwner;
}

function setConfig(NFTUpgradeableConfig memory config_) public virtual ov…
  erride {
    require(owner() == msg.sender);

    config = config_;
}

// NFTPackage.sol
function setPackage(NFTPackageConfig memory package_) public virtual over…
  ride {
    require(owner() == msg.sender);
    require(0 == tokenCount());

    packages[package_.name] = package_;
}
```

RECOMMENDATION

Consider using the modifier for code and logic consistency.

```
// NFTUpgradeable.sol
modifier onlyOwner() {
    require(owner() == msg.sender);
    _;
}

function transferOwnership(address newOwner) public virtual override only…
  Owner {
    require(newOwner != address(0));
```

```
    _owner = newOwner;
}

function setConfig(NFTUpgradeableConfig memory config_) public virtual ov…
  erride onlyOwner {
    config = config_;
}

// NFTPackage.sol
function setPackage(NFTPackageConfig memory package_) public virtual over…
  ride onlyOwner {
    require(0 == tokenCount());

    packages[package_.name] = package_;
}
```

UPDATES

- *Jan 18, 2022*: This issue has been acknowledged by the OP3N team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jan 19, 2022*	Public Report	Verichains Lab

Table 2. Report versions history