verichains

SECURITY AUDIT OF

NF3 MARKETPLACE SMART CONTRACT

Public Report

Oct 24, 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 was prepared by Verichains Lab on Oct 24, 2022. We would like to thank the NF3Labs 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 NF3 Marketplace Smart Contract. 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.

NF3Labs fixed all the issues according to Verichains's private report

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About NF3 Marketplace Smart Contract

NF3 Marketplace Smart Contract is an NFT marketplace where users can buy, sell and swap their assets (Fungible Tokens and Non-Fungible Tokens). It provides users the ability to make gas-free listing by letting them sign their order transaction offline, only the Taker (the one who fulfill the order) need to pay the gas fees for processing the on-chain transaction.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the NF3 Marketplace Smart Contract.

It was conducted on commit `1d7849d18ac7f84818590d1510acadafcf0c8eca` from git repository *https://github.com/NF3Labs/contracts-V2*.

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

SHA256 Sum	File
2a628763f889bfde82d7ced3d31529ae76ba6b05b0dafc9a967d16ee5e4322ec	contracts/lib/Utils.sol
8bd220750e1f5e06517a8c975712178f3e932c712e526873f4e93d32c0fa67bd	contracts/NF3Market.sol
412aa7027e8f8746b7bcd789a5cad6162f79b9e499c83d29e9a4908c7060d0e2	contracts/NF3Proxy.sol
ccc85b5dd3eaebb81f4078dbbb6033f60ee80ef0935dd896077084612cfcad38	contracts/PositionToken.sol
4c3cd6980672469d62f72356682c566a08428b53715226b037effde51e4c3bfb	contracts/Reserve.sol
aa1ef47f7fe679c897661e0f65f07fe134bb47ff6e505cb9f17f7ef8e48d741c	contracts/Swap.sol
760ac12fcc2387c1d986066500b920e7589f98438529423e3b572c250b0101bd	contracts/Vault.sol
c1168b76a375200d7bee9387eaaa3e95111562ab7f1a589df5bfeff9e0445d21	contracts/Whitelist.sol
173b634242eda9094b9ed1e798163da20de1985615db423a1a74dca1f15e806d	utils/DataTypes.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 were 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

NF3 Marketplace Smart Contract is an NFT marketplace where users can buy, sell and swap their assets (Fungible Tokens and Non-Fungible Tokens). It provides users the ability to make gas-free listing by letting them sign their order transaction offline, only the Taker (the one who fulfill the order) need to pay the gas fees for processing the on-chain transaction.

Users can list their assets for swapping, give swap offers, make Collection Offer and make Reserve Swap (reserve the item by paying collateral and pay remaining amount later within pay duration).

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of NF3 Marketplace Smart Contract.

NF3Labs fixed all the issues according to Verichains's private report in commit `0e1d0944cdd3fe36863593bd24be9c094bc64fc6`.

2.2.1. Seller can set wrong type in `_reserveInfo.remaining` to stop buyer from `payRemains` and steal the collateral CRITICAL

Affected files:

- contracts/Reserve.sol

When listing for a `reserve` order, seller can set wrong type in `ReserveInfo.remaining` but correct in `ReserveInfo.deposit`. As a result, buyer can make `reserveDeposit` (the buyer's deposit assets are sent to the seller in this step) but not `payRemains` (`transferAssets` call wrong ERC 721/1155 `safeTransferFrom` will be reverted). The seller now only needs to wait for the order to expire and claim back deposited assets to complete stealing the buyer's collateral.

```
function reserveDeposit(
    Listing calldata _listing,
    bytes memory _listingSignature,
    uint256 _reserveId,
    address _user,
    uint256 _value
) external override onlyMarket {
    // Check the signature, nonce and expiration.
    _listing.verifyListingSignature(_listingSignature);

    ISwap(swapAddress).checkNonce(
        _listing.owner,
        _listing.nonce,
```

```
        Status.AVAILABLE
    );

    checkExpiration(_listing.timePeriod);

    // Seller should not buy his own listing.
    _listing.owner.notItemOwner(_user);

    ReserveInfo memory reserveInfo = reserveExists(
        _listing.reserves,
        _reserveId
    );

    // Check the Eth value.
    reserveInfo.deposit.checkEthAmount(_value);

    // Mint the position token with listing & reserve info.
    IPositionToken(positionTokenAddress).mint(
        _listing.listingAssets,
        reserveInfo,
        _listing.owner,
        _user
    );

    // Exchange the assets.
    IVault(vaultAddress).transferAssets(
        reserveInfo.deposit,
        _user,
        _listing.owner,
        true
    );

    IVault(vaultAddress).transferAssets(
        _listing.listingAssets,
        _listing.owner,
        vaultAddress,
        false
    );

    // Update the nonce.
    ISwap(swapAddress).setNonce(
        _listing.owner,
        _listing.nonce,
        Status.RESERVED
    );

    emit ReserveDeposited(_listing, reserveInfo, _reserveId, _user);
}
function payRemains(
```

```
    Listing calldata _listing,
    bytes memory _listingSignature,
    ReserveInfo calldata _reserveInfo,
    uint256 _positionTokenId,
    address _user,
    uint256 _value
) external override onlyMarket {
    ...
    bytes32 dataHash = IPositionToken(positionTokenAddress).dataHash(
        _positionTokenId
    );

    // Check if the position token is of the correct listing.
    checkPositionTokenHash(
        dataHash,
        _listing.listingAssets.getPostitionTokenDataHash(
            _reserveInfo,
            _listing.owner
        )
    );

    // Check if reservation has expired.
    checkExpiration(
        _reserveInfo.duration +
        IPositionToken(positionTokenAddress).startTime(_positionTokenId)
    );

    // Check the Eth value.
    _reserveInfo.remaining.checkEthAmount(_value);

    // Exchange the assets.
    IVault(vaultAddress).transferAssets(
        _reserveInfo.remaining,
        _user,
        _listing.owner,
        true
    );

    IVault(vaultAddress).transferAssets(
        _listing.listingAssets,
        vaultAddress,
        _user,
        false
    );

    // Burn the position token.
    IPositionToken(positionTokenAddress).burn(_positionTokenId);

    // Update the listing nonce.
    ISwap(swapAddress).setNonce(
```

```
        _listing.owner,
        _listing.nonce,
        Status.EXHAUSTED
    );

    emit RemainsPaid(_listing, _reserveInfo, _positionTokenId, _user);
}

function transferAssets(
    Assets calldata _assets,
    address _from,
    address _to,
    bool _allowEth
) external override onlyApproved {
    ...
    for (i = 0; i < _assets.tokens.length; i++) {
        if (_assets.types[i] == 0) {
            IERC721(_assets.tokens[i]).safeTransferFrom(
                _from,
                _to,
                _assets.tokenIds[i]
            );
        } else if (_assets.types[i] == 1) {
            IERC1155(_assets.tokens[i]).safeTransferFrom(
                _from,
                _to,
                _assets.tokenIds[i],
                1,
                ""
            );
        } else {
            revert VaultError(VaultErrorCodes.INVALID_ASSET_TYPE);
        }
    }
    ...
}
```

RECOMMENDATION

The contract needs a mapping for address to ERC types instead of believing in the seller.

UPDATES

- *Oct 24, 2022*: This issue has been acknowledged and fixed by the NF3Labs team.

2.2.2. Possible fund lost LOW

Affected files:

- contracts/NF3Market.sol

Contract `NF3Market` enables native tokens (ETH, BNB...) receiving in the `receive` function but does not use the fund. If anyone sends native tokens to the contract by mistake the fund will be locked and the contract have to be upgraded to add withdraw logic to recover the fund.

```
receive() external payable {}
```

RECOMMENDATION

Consider removing `receive` function in the `NF3Market` contract.

UPDATES

- *Oct 24, 2022*: This issue has been acknowledged and fixed by the NF3Labs team.

2.3. Additional notes and recommendations

2.3.1. Using EIP-1967 for proxy pattern INFORMATIVE

Consider using EIP-1967 for `NF3Proxy` to help other tools (like block explorer) and end users to interact with the contract easier.

UPDATES

- *Oct 24, 2022*: This issue has been acknowledged by the NF3Labs team.

2.3.2. `_newImplementation` should be required to not equal zero address INFORMATIVE

Affected files:

- contracts/NF3Proxy.sol

`_newImplementation` should be required to not equal zero address in `_upgradeTo` function.

```
function _upgradeTo(address _newImplementation) internal {
    address currentImplementation = _implementation();
    require(currentImplementation != _newImplementation);
    _setImplementation(_newImplementation);
}
```

RECOMMENDATION

```
function _upgradeTo(address _newImplementation) internal {
    require(_newImplementation != address(0), "New implementation should not be zero
address");
    address currentImplementation = _implementation();
    require(currentImplementation != _newImplementation);
    _setImplementation(_newImplementation);
}
```

UPDATES

- *Oct 24, 2022*: This issue has been acknowledged and fixed by the NF3Labs team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Sep 29, 2022*	Private Report	Verichains Lab
1.1	*Sep 30, 2022*	Public Report	Verichains Lab
1.2	*Oct 24, 2022*	Public Report	Verichains Lab

Table 2. Report versions history