Armors Labs

Scape Store (NFT Marketplace)

Smart Contract Audit

Scape Store (NFT Marketplace) Audit Summary

Project name : Scape Store (NFT Marketplace) Contract

Project address: None

Code URL : https://github.com/blocklords/seascape-smartcontracts/blob/nft-market/contracts/nft_marketplace/NftMarket.sol

Commit : 4e157117ddc81f3ad29cd7b5d879f44c36e61009

Project target : Scape Store (NFT Marketplace) Contract Audit

Blockchain : Binance Smart Chain（BSC）

Test result : PASSED

Audit Info

Audit NO : 0X202106220016

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

Scape Store (NFT Marketplace) Audit

The Scape Store (NFT Marketplace) team asked us to review and audit their Scape Store (NFT Marketplace) contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
Scape Store (NFT Marketplace) Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-06-22

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Scape Store (NFT Marketplace) contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to Armors Labs at the time of issuance of this report (" information provided " for short). Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused.)

Audited target file

file	md5
NftMarket.sol	ab15bd29527b80bcf16c57b7c6a99606

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe

Vulnerability	status
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

NftMarket.sol

```
pragma solidity ^0.6.7;
pragma experimental ABIEncoderV2;

import "./../openzeppelin/contracts/token/ERC20/IERC20.sol";
import "./../openzeppelin/contracts/math/SafeMath.sol";
import "./../openzeppelin/contracts/token/ERC721/IERC721.sol";
import "./../openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
import "./../openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "./../openzeppelin/contracts/access/Ownable.sol";
import "./../seascape_nft/SeascapeNft.sol";
import "./ReentrancyGuard.sol";

/// @title Nft Market is a trading platform on seascape network allowing to buy and sell Nfts
/// @author Nejc Schneider
contract NftMarket is IERC721Receiver,  ReentrancyGuard, Ownable {
    using SafeERC20 for IERC20;
    using SafeMath for uint256;

    /// @notice individual sale related data
    struct SalesObject {
        uint256 id;               // sales ID
        uint256 tokenId;          // token unique id
        address nft;              // nft address
        address currency;         // currency address
        address payable seller;   // seller address
        address payable buyer;    // buyer address
        uint256 startTime;        // timestamp when the sale starts
        uint256 price;            // nft price
        uint8 status;             // 2 = sale canceled, 1 = sold, 0 = for sale
    }

    /// @dev keep count of SalesObject amount
    uint256 public salesAmount;

    /// @dev store sales objects.
    /// @param nft token address => (nft id => salesObject)
    mapping(address => mapping(uint256 => SalesObject)) salesObjects; // store sales in a mapping

    /// @dev supported ERC721 and ERC20 contracts
    mapping(address => bool) public supportedNft;
    mapping(address => bool) public supportedCurrency;

    /// @notice enable/disable trading
    bool public salesEnabled;

    /// @dev fee rate and fee reciever. feeAmount = (feeRate / 1000) * price
    uint256 public feeRate;
    address payable feeReceiver;

    event Buy(
        uint256 indexed id,
        uint256 tokenId,
        address buyer,
        uint256 price,
        uint256 tipsFee,
        address currency
    );

    event Sell(
        uint256 indexed id,
        uint256 tokenId,
        address nft,
        address currency,
        address seller,
        address buyer,
        uint256 startTime,
        uint256 price
    );

    event SaleCanceled(uint256 indexed id, uint256 tokenId);
    event NftReceived(address operator, address from, uint256 tokenId, bytes data);

    /// @dev set fee reciever address and fee rate
    /// @param _feeReceiver fee receiving address
    /// @param _feeRate fee amount
    constructor(address payable _feeReceiver, uint256 _feeRate) public {
        feeReceiver = _feeReceiver;
        feeRate = _feeRate;
        initReentrancyStatus();
    }

    //--------------------------------------------------
    // External methods
    //--------------------------------------------------

    /// @notice enable/disable sales
    /// @param _salesEnabled set sales to true/false
    function enableSales(bool _salesEnabled) external onlyOwner { salesEnabled = _salesEnabled; }

    /// @notice add supported nft token
    /// @param _nftAddress ERC721 contract address
    function addSupportedNft(address _nftAddress) external onlyOwner {
        require(_nftAddress != address(0x0), "invalid address");
        supportedNft[_nftAddress] = true;
    }

    /// @notice disable supported nft token
    /// @param _nftAddress ERC721 contract address
    function removeSupportedNft(address _nftAddress) external onlyOwner {
        require(_nftAddress != address(0x0), "invalid address");
        supportedNft[_nftAddress] = false;
    }

    /// @notice add supported currency token
    /// @param _currencyAddress ERC20 contract address
    function addSupportedCurrency(address _currencyAddress) external onlyOwner {
        require(_currencyAddress != address(0x0), "invalid address");
        require(!supportedCurrency[_currencyAddress], "currency already supported");
        supportedCurrency[_currencyAddress] = true;
    }

    /// @notice disable supported currency token
    /// @param _currencyAddress ERC20 contract address
    function removeSupportedCurrency(address _currencyAddress) external onlyOwner {
        require(_currencyAddress != address(0x0), "invalid address");
        require(supportedCurrency[_currencyAddress], "currency already removed");
        supportedCurrency[_currencyAddress] = false;
    }
    /// @notice change fee receiver address
    /// @param _walletAddress address of the new fee receiver
    function setFeeReceiver(address payable _walletAddress) external onlyOwner {
        require(_walletAddress != address(0x0), "invalid address");
        feeReceiver = _walletAddress;
    }

    /// @notice change fee rate
    /// @param _rate amount value. Actual rate in percent = _rate / 10
    function setFeeRate(uint256 _rate) external onlyOwner {
        require(_rate <= 100, "Rate should be bellow 100 (10%)");
        feeRate = _rate;
    }

    /// @notice returns sales amount
    /// @return total amount of sales objects
    function getSalesAmount() external view returns(uint) { return salesAmount; }

    //--------------------------------------------------
    // Public methods
    //--------------------------------------------------

    /// @notice cancel nft sale
    /// @param _tokenId nft unique ID
    /// @param _nftAddress nft token address
    function cancelSell(uint _tokenId, address _nftAddress) public nonReentrant {
        SalesObject storage obj = salesObjects[_nftAddress][_tokenId];
        require(obj.status == 0, "status: sold or canceled");
        require(obj.seller == msg.sender, "seller not nft owner");
        require(salesEnabled, "sales are closed");
        obj.status = 2;
        IERC721 nft = IERC721(obj.nft);
        nft.safeTransferFrom(address(this), obj.seller, obj.tokenId);
        emit SaleCanceled(_tokenId, obj.tokenId);
    }

    /// @notice put nft for sale
    /// @param _tokenId nft unique ID
    /// @param _price required price to pay by buyer. Seller receives less: price - fees
    /// @param _nftAddress nft token address
    /// @param _currency currency token address
    /// @return salesAmount total amount of sales
    function sell(uint256 _tokenId, uint256 _price, address _nftAddress, address _currency)
        public
        nonReentrant
        returns(uint)
    {
        require(_nftAddress != address(0x0), "invalid nft address");
        require(_tokenId != 0, "invalid nft token");
        require(salesEnabled, "sales are closed");
        require(supportedNft[_nftAddress], "nft address unsupported");
        require(supportedCurrency[_currency], "currency not supported");
        IERC721(_nftAddress).safeTransferFrom(msg.sender, address(this), _tokenId);

        salesAmount++;

        salesObjects[_nftAddress][_tokenId] = SalesObject(
            salesAmount,
            _tokenId,
            _nftAddress,
            _currency,
            msg.sender,
            address(0x0),
            now,
            _price,
            0
        );

        emit Sell(
            salesAmount,
            _tokenId,
            _nftAddress,
            _currency,
            msg.sender,
            address(0x0),
            now,
            _price
        );

        return salesAmount;
    }

    /// @dev encrypt token data
    function onERC721Received(
        address operator,
        address from,
        uint256 tokenId,
        bytes memory data
    )
        public
        override
        returns (bytes4)
    {
        //only receive the _nft staff
        if (address(this) != operator) {
            //invalid from nft
            return 0;
        }

        //success
        emit NftReceived(operator, from, tokenId, data);
        return bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"));
    }

    /// @notice buy nft
    /// @param _tokenId nft unique ID
    /// @param _nftAddress nft token address
    /// @param _currency currency token address
    function buy(uint _tokenId, address _nftAddress, address _currency)
        public
        nonReentrant
        payable
    {
        SalesObject storage obj = salesObjects[_nftAddress][_tokenId];
        require(obj.status == 0, "status: sold or canceled");
        require(obj.startTime <= now, "not yet for sale");
        require(salesEnabled, "sales are closed");
        require(msg.sender != obj.seller, "cant buy from yourself");

        require(obj.currency == _currency, "must pay same currency as sold");
        uint256 price = this.getSalesPrice(_tokenId, _nftAddress);
        uint256 tipsFee = price.mul(feeRate).div(1000);
        uint256 purchase = price.sub(tipsFee);

        if (obj.currency == address(0x0)) {
            require (msg.value >= price, "your price is too low");
            uint256 returnBack = msg.value.sub(price);
            if (returnBack > 0)
                msg.sender.transfer(returnBack);
            if (tipsFee > 0)
                feeReceiver.transfer(tipsFee);
            obj.seller.transfer(purchase);
        } else {
            IERC20(obj.currency).safeTransferFrom(msg.sender, feeReceiver, tipsFee);
            IERC20(obj.currency).safeTransferFrom(msg.sender, obj.seller, purchase);
        }

        IERC721 nft = IERC721(obj.nft);
        nft.safeTransferFrom(address(this), msg.sender, obj.tokenId);
        obj.buyer = msg.sender;

        obj.status = 1;
        emit Buy(obj.id, obj.tokenId, msg.sender, price, tipsFee, obj.currency);
    }

    /// @dev fetch sale object at nftId and nftAddress
    /// @param _tokenId unique nft ID
    /// @param _nftAddress nft token address
    /// @return SalesObject at given index
    function getSales(uint _tokenId, address _nftAddress)
        public
        view
        returns(SalesObject memory)
    {
        return salesObjects[_nftAddress][_tokenId];
    }

    /// @dev returns the price of sale
    /// @param _tokenId nft unique ID
    /// @param _nftAddress nft token address
    /// @return obj.price price of corresponding sale
    function getSalesPrice(uint _tokenId, address _nftAddress) public view returns (uint256) {
        SalesObject storage obj = salesObjects[_nftAddress][_tokenId];
        return obj.price;
    }

}
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**
  ```
  PASSED!
  ```
- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
no.

Unexpected Blockchain Currency

- **Description:**
Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulnerabilities in smart contracts as will be discussed in this section.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io