verichains

SECURITY AUDIT OF

MONES SMART CONTRACTS

Public Report

Apr 04, 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 Apr 04, 2022. We would like to thank the Mones 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 Mones 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 Mones Smart Contracts

Mones is a Hero-collecting RPG where you collect over 200 hero characters, turn them into skilled warriors through training then fight in PvE/PvP battles to take over the kingdom.

1.2. Audit scope

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

It was conducted on commit 3b916e72199694a18b1bd322591854d41cfd92e8 from git repository *https://git.xantus.network/mones/smart-contracts*.

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

SHA256 Sum	File
05a812e9131a53e00d642ec35bede0187a1b208e27280b7ee04ba79d5320556b	**Diamond.sol**
c94ab996604bb193bb0012db22166a705eab36043439efc73b9e6a7cb8e20a1a	**AccessControlEnumerableFacet.sol**
e7a523fd3787594150058d25aef9303d4c81f6e08e9bcab38847ff5c232013b0	**AccessControlFacet.sol**
4b7aed22655b59e76ede32d5a70296b1bcea0abbcea21d3e6bd9c5808fc9f434	**DiamondCutFacet.sol**
66530a20e8ab3e8ee9b23939733964a8f03dbe5ee7980a57b958997dbb35c09d	**DiamondLoupeFacet.sol**
1961b34399d6132669966d89cebe78d212a5d2feaff30051c19250cba5023b31	**DiamondLib.sol**
9df62ed40d97e8ca2b9d843ff0833b22162e797b8f100d12d11c1e0c1fdef389	**StorageSlots.sol**
4b61cf7cbebc7a2696f39b05cdd8fc102f6e9e0d7c7199cd9ff93212a81d312f	**ChannelDataFacet.sol**
260ca95a0d4c4fc4908fd0816a21cbbc3d34faf289842422ef0e8a85ab32ff45	**Constant.sol**
4cb4b5baeea04d7a975cd30cb92cfcab2b9341c314e80050450d147849153b74	**ExchangeNFTFacet.sol**
7c87bf214d9fda55dec34bdd59be2afd16eb06d92498123452535137fff04273	**Marketplace.sol**
0f0f35602bb9a8c730b99e93cfb50a4ae7605916a8279e567156d91565b6849c	**HeroERC721.sol**
6236c7c006fa71bcce34ad370eee7d533790c55eb822199813b4c638ccd5641d	**MonesERC721.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

The Mones Smart Contracts was written in Solidity language. The source code was written based on OpenZeppelin's library.

There are two main contracts in the audit scope. They are HeroERC721 contract and MonesMarketplace contract.

2.1.1. HeroERC721 contract

This is the ERC721 NFT contract in the Mones Smart Contracts, which extends ERC721, ERC721URIStorage and AccessControl contracts. AccessControl allows the contract to implement role-based access control mechanisms which adds token owner (contract deployer) DEFAULT_ADMIN_ROLE and MINTER_ROLE roles, DEFAULT_ADMIN_ROLE then can set any roles for anyone at anytime. The contract implements safeMint public function which allows only MINTER_ROLE to mint new NFTs.

Only DEFAULT_ADMIN_ROLE can set up genesis heroes. Then, MINTER_ROLE can safeMint heroes (NFTs) with most stats (name, element, class, rarity, star) based on genesis heroes, only level and enhancement are set to 1 and 0 respectively by default. Rarity, star, level and enhancement can be updated by MINTER_ROLE at any time.

2.1.2. MonesMarketplace contract

This is the marketplace contract in the Mones Smart Contracts, which extends ERC721Holder, ERC1155Holder, Ownable and ReentrancyGuard contracts. With Ownable, by default, Contract Owner is contract deployer, but he can transfer ownership to another address at any time.

The marketplace has a _platformFee percentage (set when deploying contract) which will be charged on every transaction. Contract Owner first adds some collections, each collection accept assets from one NFT contract. Each collection has a creatorFee (can be zero) percentage and belong to a creator. Sellers can list their assets (NFTs) to the added collections. They can choose either ERC20 tokens or native tokens to use for the payment. When a transaction is completed, an amount of fee calculated by creatorFee plus platformFee (must not be greater than 10%) will be taken, creatorFee is paid to collection's creator and platformFee is paid to the Contract Owner. Buyers can choose to buy the assets with listed price or send offers which their own price to the sellers. The sellers can accept the offers if they want.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Mones Smart Contracts.

Mones fixed the code, according to Verichains's private report, in commit cd3ef7db844d1fecb4d09520d13d75891109793a.

2.2.1. Marketplace.sol - Missing owner check CRITICAL

The modifyAsk function is missing owner check so anyone can cancel the ask and steal NFT from the marketplace.

```
function modifyAsk(
    uint256 _askId,
    bool _status,
    uint256 _unitPrice
) external nonReentrant {
    Ask storage asking = _asks[_askId];
    require(asking.status, "Ask: This ask already canceled!");
    require(asking.amount > 0, "Ask: This ask already done!");
    asking.status = _status;
    if (!_status) {
        _transferAsset(asking.nftAddress, asking.nftId, asking.amount, ad…
  dress(this), msg.sender, "0x");
    } else {
        asking.unitPrice = _unitPrice;
    }
    emit AskUpdated(_askId, asking.status, asking.amount, asking.unitPric…
  e);
}
```

RECOMMENDATION

Adding owner check to modifyAsk function.

```
function modifyAsk(
    uint256 _askId,
    bool _status,
    uint256 _unitPrice
) external nonReentrant {
    Ask storage asking = _asks[_askId];

    require(asking.creator == msg.sender, "Offer: Only Asking Creator can…
```

```
    modify the offer!");

    require(asking.status, "Ask: This ask already canceled!");
    require(asking.amount > 0, "Ask: This ask already done!");
    asking.status = _status;
    if (!_status) {
        _transferAsset(asking.nftAddress, asking.nftId, asking.amount, ad…
  dress(this), msg.sender, "0x");
    } else {
        asking.unitPrice = _unitPrice;
    }
    emit AskUpdated(_askId, asking.status, asking.amount, asking.unitPric…
  e);
}
```

UPDATES

- *Mar 31, 2022*: This issue has been acknowledged and fixed by the Mones.

2.2.2. Marketplace.sol - Front running in buyAssets CRITICAL

Attackers can list an item by listAssets with low price and wait for a user buy it with buyAssets. While waiting, attackers listen for pending transactions and when a user buy that item, they send a transaction update price modifyAsk with higher gas price (higher gas price transaction is usually mined first) than the buy transaction and change price of the item to a higher number. The result is user buy the item with more money than he saw in the marketplace.

For example: Attacker put an item with 1 USDT in marketplace, user see it and make a transaction to buy it with 1 USDT. Attacker listen to pending transactions and know that someone is buying the item with 1 USDT and send a transaction update price to 1000 USDT with higher gas price and get mined before the buy transaction. The result is user lost 1000 USDT to attacker for that item.

```
function modifyAsk(
    uint256 _askId,
    bool _status,
    uint256 _unitPrice
) external nonReentrant {
    Ask storage asking = _asks[_askId];
    require(asking.status, "Ask: This ask already canceled!");
    require(asking.amount > 0, "Ask: This ask already done!");
    asking.status = _status;
    if (!_status) {
```

```
        _transferAsset(asking.nftAddress, asking.nftId, asking.amount, ad…
  dress(this), msg.sender, "0x");
    } else {
        asking.unitPrice = _unitPrice;
    }
    emit AskUpdated(_askId, asking.status, asking.amount, asking.unitPric…
  e);
}

function buyAssets (
    uint256[] memory _askIds,
    uint256[] memory _amounts
) external payable nonReentrant {
    uint256 price;
    uint256 feeForCreator;
    uint256 feeForPlatform;
    uint256 netPrice;
    uint256 rawValue;
    for (uint256 i = 0; i < _askIds.length; i++) {
        Ask storage asking = _asks[_askIds[i]];
        require(asking.status, "Ask: was closed!");
        require(_amounts[i] != 0, "Ask: Zero quantity!");
        require(asking.amount >= _amounts[i], "Ask: quantity not availalb…
  le!");
        price = asking.unitPrice * _amounts[i];
        feeForCreator = price * _collections[asking.nftAddress].creatorFe…
  e / TOTAL_PERCENT;
        feeForPlatform = price * _platformFee / TOTAL_PERCENT;
        netPrice = price - (feeForCreator + feeForPlatform);
        if (asking.paymentAddress == address(0)) {
            rawValue += price;
        } else {
            IERC20(asking.paymentAddress).safeTransferFrom(msg.sender, ad…
  dress(this), price);
        }
        _payout(asking.paymentAddress, asking.creator, netPrice);
        _revenue[_collections[asking.nftAddress].creator][asking.paymentA…
  ddress] += feeForCreator;
        _revenue[platformAddress][asking.paymentAddress] += feeForPlatfor…
  m;
        _transferAsset(asking.nftAddress, asking.nftId, _amounts[i], addr…
  ess(this), msg.sender, "0x");
```

```
        // event AssetBought(uint256 indexed listId, address indexed buye…
  r, address indexed nftAddress, uint256 indexed nftId, uint256 indexed a…
  mount, address indexed paymentAddress, uint256 indexed price);
        emit AssetBought(_askIds[i], msg.sender, asking.nftAddress, askin…
  g.nftId, _amounts[i], asking.paymentAddress, price);
        asking.amount -= _amounts[i];
        // event AskUpdated(uint256 indexed listId, bool status, uint256 …
  _amount, uint256 _price);
        emit AskUpdated(_askIds[i], asking.status, asking.amount, asking.…
  unitPrice);
    }
    if (msg.value >= rawValue) {
        _payout(address(0), msg.sender, (msg.value - rawValue));
    } else {
        revert("BuyAssets: Not enough amount");
    }
}
```

RECOMMENDATION

Add _unitPrice parameter to buyAssets to revert when asking price is higher than the price user want to buy.

```
function buyAssets (
    uint256[] memory _askIds,
    uint256[] memory _amounts,
    uint256[] memory _unitPrice
) external payable nonReentrant {
    uint256 price;
    uint256 feeForCreator;
    uint256 feeForPlatform;
    uint256 netPrice;
    uint256 rawValue;
    for (uint256 i = 0; i < _askIds.length; i++) {
        Ask storage asking = _asks[_askIds[i]];
        require(asking.status, "Ask: was closed!");
        require(_amounts[i] != 0, "Ask: Zero quantity!");
        require(asking.amount >= _amounts[i], "Ask: quantity not availalb…
  le!");

        require(asking.unitPrice <= _unitPrice[i], "Ask: asking price is …
  higher than buy price!");
```

```
        price = asking.unitPrice * _amounts[i];
        feeForCreator = price * _collections[asking.nftAddress].creatorFe…
  e / TOTAL_PERCENT;
        feeForPlatform = price * _platformFee / TOTAL_PERCENT;
        netPrice = price - (feeForCreator + feeForPlatform);
        if (asking.paymentAddress == address(0)) {
            rawValue += price;
        } else {
            IERC20(asking.paymentAddress).safeTransferFrom(msg.sender, ad…
  dress(this), price);
        }
        _payout(asking.paymentAddress, asking.creator, netPrice);
        _revenue[_collections[asking.nftAddress].creator][asking.paymentA…
  ddress] += feeForCreator;
        _revenue[platformAddress][asking.paymentAddress] += feeForPlatfor…
  m;
        _transferAsset(asking.nftAddress, asking.nftId, _amounts[i], addr…
  ess(this), msg.sender, "0x");
        // event AssetBought(uint256 indexed listId, address indexed buye…
  r, address indexed nftAddress, uint256 indexed nftId, uint256 indexed a…
  mount, address indexed paymentAddress, uint256 indexed price);
        emit AssetBought(_askIds[i], msg.sender, asking.nftAddress, askin…
  g.nftId, _amounts[i], asking.paymentAddress, price);
        asking.amount -= _amounts[i];
        // event AskUpdated(uint256 indexed listId, bool status, uint256 …
  _amount, uint256 _price);
        emit AskUpdated(_askIds[i], asking.status, asking.amount, asking.…
  unitPrice);
    }
    if (msg.value >= rawValue) {
        _payout(address(0), msg.sender, (msg.value - rawValue));
    } else {
        revert("BuyAssets: Not enough amount");
    }
}
```

UPDATES

- *Mar 31, 2022*: This issue has been acknowledged and fixed by the Mones.

2.3. Additional notes and recommendations

2.3.1. Marketplace.sol - Typos INFORMATIVE

There are some typos in the contracts.

require(_amount > 0, "Offer: Zero Quantiy"); should be require(_amount > 0, "Offer: Zero Quantity");.

require(asking.amount >= _amounts[i], "Ask: quantity not availalble!"); should be require(asking.amount >= _amounts[i], "Ask: insufficient quantity!");.

RECOMMENDATION

Fixing the typo.

UPDATES

- *Mar 31, 2022*: This issue has been acknowledged and fixed by the Mones.

2.3.2. Marketplace.sol - Usage of constant and immutable INFORMATIVE

The TOTAL_MAX_FEE and TOTAL_PERCENT variables are not changed in the contract, so it's better to use constant instead of immutable.

The platformAddress variable is only set to zero one time in the constructor, so it's better to use a constant variable.

Read more here: *https://docs.soliditylang.org/en/v0.8.12/contracts.html#constant-and-immutable-state-variables*

```
uint256 private immutable TOTAL_MAX_FEE = 1000;
uint256 private immutable TOTAL_PERCENT = 10000;

address public platformAddress;

constructor(
    uint256 _platformFeePercent
) {
    _platformFee = _platformFeePercent;
    platformAddress = address(0);
    _isPaymentAccepted[address(0)] = true;
}
```

RECOMMENDATION

Using constant instead of immutable for TOTAL_MAX_FEE and TOTAL_PERCENT.

Using constant for platformAddress.

```
uint256 private constant TOTAL_MAX_FEE = 1000;
uint256 private constant TOTAL_PERCENT = 10000;

address public constant platformAddress = address(0);

constructor(
    uint256 _platformFeePercent
) {
    _platformFee = _platformFeePercent;
    _isPaymentAccepted[address(0)] = true;
}
```

UPDATES

- *Mar 31, 2022*: This issue has been acknowledged and fixed by the Mones.

2.3.3. Marketplace.sol - paymentAddress can be contract which has transfer fee INFORMATIVE

In buyAssets function, if the paymentAddress is a contract which has transfer fee, the amount of tokens the contract received from msg.sender can be less than price (due to the fee), so the contract will receive fewer tokens which causing not enough tokens to pay the revenue. The paymentAddress must be allowed by Contract Owner to be used so please check carefully that paymentAddress will not take any transfer fee.

```
function buyAssets (
    uint256[] memory _askIds,
    uint256[] memory _amounts
) external payable nonReentrant {
    ...
    if (asking.paymentAddress == address(0)) {
        rawValue += price;
    } else {
        IERC20(asking.paymentAddress).safeTransferFrom(msg.sender, addres…
  s(this), price);
    }
    _payout(asking.paymentAddress, asking.creator, netPrice);
    ...
}
```

```
function _payout(
    address payment,
    address to,
    uint256 amount
) internal {
    if (payment == address(0)) {
        payable(to).transfer(amount);
    } else {
        IERC20(payment).safeTransfer(to, amount);
    }
}
```

UPDATES

- *Mar 31, 2022*: This issue has been acknowledged by the Mones.

2.3.4. Marketplace.sol - Using > instead of >= INFORMATIVE

In buyAssets function, if msg.value is equal to rawValue, the contract will transfer 0 native tokens to msg.sender and it's only wasting gas.

```
function buyAssets (
    uint256[] memory _askIds,
    uint256[] memory _amounts
) external payable nonReentrant {
    ...
    if (msg.value >= rawValue) {
        _payout(address(0), msg.sender, (msg.value - rawValue));
    } else {
        revert("BuyAssets: Not enough amount");
    }
}

function _payout(
    address payment,
    address to,
    uint256 amount
) internal {
    if (payment == address(0)) {
        payable(to).transfer(amount);
    } else {
        IERC20(payment).safeTransfer(to, amount);
```

```
    }
}
```

RECOMMENDATION

Using > instead of >=.

```
function buyAssets (
    uint256[] memory _askIds,
    uint256[] memory _amounts
) external payable nonReentrant {
    ...
    require(msg.value >= rawValue, "BuyAssets: Not enough amount of payme…
  nt!");
    if (msg.value > rawValue) {
        _payout(address(0), msg.sender, (msg.value - rawValue));
    }
}
```

UPDATES

- *Mar 31, 2022*: This issue has been acknowledged and fixed by the Mones.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Mar 29, 2022*	Private Report	Verichains Lab
1.1	*Mar 31, 2022*	Public Report	Verichains Lab
1.2	*Apr 04, 2022*	Public Report	Verichains Lab

Table 2. Report versions history