verichains

SECURITY AUDIT OF

HOUSE OF KIBAA SMART CONTRACTS

Public Report

Apr 22, 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 22, 2022. We would like to thank the House of Kibaa 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 House of Kibaa 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 House of Kibaa Smart Contracts

House of Kibaa is the innovation hub for all things NFTs. Our blend of creative wizardry and expertise will let you realize the boundless potential of the coming NFT revolution. Whether you're a content creator, NFT collector, or tech evangelist, our savviness has you covered.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of House of Kibaa Smart Contracts. It was conducted on the source code provided by the House of Kibaa team.

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

SHA256 Sum	File
96dbde0b4af5242dadac265f7e5c2b036beba712d37835350001b2622b6fa8d5	**LandNFT.sol**
1cbea7b11f0d5aebee772baac17ad8fa72e0a0915ebaf29a9f819ea8100743a3	**PrivatePublicSale.sol**
bd11189be09dd755afe810b145a393eeb8eaaf6565506291d47611c5ad48306a	**PocketDimension.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 House of Kibaa Smart Contracts was written in Solidity language, with the required version to be ^0.8.0.

2.1.1. LandNFT contract

LandNFT contract is an ERC721 contract which extends ERC721, ReentrancyGuard, Pausable and Ownable contract. With Ownable, by default, Token Owner is contract deployer, but he can transfer ownership to another address at any time. The owner may set saleActive value. The users only transfer NFT tokens when saleActive value is false. The saleContractAdddress account that was set by the owner may mint new tokens.

The contract implements the withdrawTokens function allows the owner to withdraw any ERC20 in this contract to a withdrawAdress. The contract also implements royaltyInfo function that was declared in the IERC2981 interface. This function allows the House of Kibaa team to create new features with this contract.

2.1.2. PrivatePublicLandSale contract

PrivatePublicLandSale is a sale contract that supports House of Kibaa team to sell the LandNFT token. The contract sells LandNFT tokens with 2 phases. In the first phase - privateSalePhase, the only users in the whitelist may buy this phase with BOTTOM_PRICE. In the second phase - publicSalePhase, anyone can purchase tokens with price drops over time.

2.1.3. PocketDimension contract

PocketDimension is an ERC721 contract which extends ERC721A contract. The contract also implementc the sale logic which allows the users to buy NFTs.

2.2. Findings

During the audit process, the audit team found some vulnerability issues in the given version of House of Kibaa Smart Contracts.

2.2.1. PrivatePublicSale.sol - User can buy NFT with price lower than the BOTTOM_PRICE value HIGH

The _getCurrentPrice function is used to calculate the price of the NFT token for sale. The price is decreased following the time with the minimum price is BOTTOM_PRICE. But at the PUBLIC_SALE_PERIOD + publicsaleStartsAt time, the user can purchase NFT with price lower than the BOTTOM_PRICE value because of the _getCurrentPrice function.

```
uint256 public constant TICK_PERIOD = 30 minutes; // Time period to decre…
  ase price
    uint256 public constant PUBLIC_SALE_PERIOD = 3 hours;   // Dutch Auct…
  ion Time Period
    uint256 public constant STARTING_PRICE = 2000000000000000000;  // 1 E…
  TH
    uint256 public constant BOTTOM_PRICE = 1000000000000000000; // 1 ETH
    uint256 public constant SALE_PRICE_STEP = 20000000000000000; // 0.2 E…
  TH

    ...

    function _getCurrentPrice() internal view returns (uint256 ){
        uint256 time = (block.timestamp);
            uint256 price = BOTTOM_PRICE;
        if(time > (PUBLIC_SALE_PERIOD + publicsaleStartsAt)) {
            return price;
        }
        uint256 timeSlot = (time-publicsaleStartsAt) / TICK_PERIOD;
        price = STARTING_PRICE - (SALE_PRICE_STEP * timeSlot);
        return price;
    }
```

Snippet 1. PrivatePublicSale.sol Issue in the `_getCurrentPrice` function

Currently, the SALE_PRICE_STEP value is 0.02 ETH. However, we think the intended value for it may be 0.2 ETH as shown in the comment.

At the PUBLIC_SALE_PERIOD + publicsaleStartsAt time, the timeSlot will be PUBLIC_SALE_PERIOD/TICK_PERIOD = 6. Therefore, the price value will be STARTING_PRICE - (SALE_PRICE_STEP * timeSlot) = 0.8 ETH lower than BOTTOM_PRICE (1 ETH).

RECOMMENDATION

We suggest changing the function like the below code:

```
function _getCurrentPrice() internal view returns (uint256 ){
        uint256 time = (block.timestamp);
            uint256 price = BOTTOM_PRICE;
        if(time > (PUBLIC_SALE_PERIOD + publicsaleStartsAt)) {
            return price;
        }
        uint256 timeSlot = (time-publicsaleStartsAt) / TICK_PERIOD;
```

```
        price = STARTING_PRICE - (SALE_PRICE_STEP * timeSlot);
        price= BOTTOM_PRICE > price ? BOTTOM_PRICE : price
        return price;
    }
```

Snippet 2. PrivatePublicSale.sol Recommend fixing in the `_getCurrentPrice` function

UPDATES

- *Mar 31, 2022*: This issue has been acknowledged and fixed by the House of Kibaa team.

2.2.2. PocketDimensionUser.sol - User can't buy enough number as the whitelist balance in mintPrivateSale function MEDIUM

```
function mintPrivateSale(uint8 landType, uint256 numTokens)
        external payable
    {
        ...
        require(_memberslist[msg.sender].exists, "Restricted access");
        require(_memberslist[msg.sender].minted + numTokens < _memberslis…
  t[msg.sender].balance, "User mint over allowed number"); // fix PPS-02
        require(landType > 0 && landType < 11, "Invalid Land type");

        require(_landTypeCounter[landType] + numTokens < MAX_PRIVATE_LAND…
  _SALE_PER_TYPE + 1, "Sold out");
        ...
```

Snippet 3. PocketDimensionUser.sol - User can't buy enough number as the whitelist balance in `mintPrivateSale` function

In the private sale phase, the user in the whitelist may buy with the value which was set by the owner. But the require statement which checks the value seems to be wrong.

For instance, if the _memberslist[msg.sender].balance value is 1, user may purchase an item. But the required statement will raise an error and revert.

UPDATES

- *Apr 22, 2022*: This issue has been acknowledged by the House of Kibaa team.

2.3. Additional notes and recommendations

2.3.1. PrivatePublicSale.sol - The numbers aren't corresponding to the comments INFORMATIVE

Some state variables in the contract aren't corresponding to the comments. They may be wrong. So we note this issue to notice the client.

```
39  uint256 public constant STARTING_PRICE = 200000000000000000;  // 1 E…
    TH
40      uint256 public constant BOTTOM_PRICE = 100000000000000000; // 1 …
    ETH
41      uint256 public constant SALE_PRICE_STEP = 20000000000000000; // 0…
    .2 ETH
```

Currently, the SALE_PRICE_STEP value is 0.02 ETH.

UPDATES

- *Mar 31, 2022*: This issue has been acknowledged and fixed by the House of Kibaa team.

2.3.2. LandNFT.sol - Redundant function in the contract INFORMATIVE

There are two functions in the contract that only update the saleContractAdddress value and have no emit events. The below code will show them:

```
function setSaleContractAddress(address _saleContractAddress) public only…
  Owner {
        saleContractAdddress = _saleContractAddress;
    }

function changeSaleContractAddress(address _saleContractAddress) public o…
  nlyOwner {
        require(_saleContractAddress != address(0),'Non Zero Address');
        // emit WithdrawAddressChanged(msg.sender, hokWithdrawAddress, _n…
  ewAddress);
        saleContractAdddress = _saleContractAddress;
    }
```

Snippet 4. LandNFT.sol Two function have the same working flow in the contract

Both of the two functions have the same action with the saleContractAdddress value. The setSaleContractAddress function may cover the changing value flow. So we suggest removing the changeSaleContractAddress function for readability. Or the team wants to keep this function,

the function should be added an emit event to make a log instead of the saleContractAdddress variable.

UPDATES

- *Mar 31, 2022*: This issue has been acknowledged and fixed by the House of Kibaa team.

2.3.3. PrivatePublicSale.sol - Redundant code in withdraw function INFORMATIVE

The console.log statements are used to support debugging in the local environment but it is useless after deploying the contract to mainnet. So we suggest removing them for gas saving and readability.

```
function withdraw(uint256 _amount) public nonReentrant {
            console.log(msg.sender);
            require(msg.sender == hokWithdrawAddress, "Not allowed");
            console.log(_amount);
            uint256 balance = address(this).balance;
            require(_amount <= balance,"Insufficient funds");
            console.log(balance);

            bool success;
            (success, ) = payable(hokWithdrawAddress).call{value: _amount…
  }('');
            require(success, 'Withdraw Failed');

            emit ContractWithdraw(msg.sender, hokWithdrawAddress, _amount…
  );
      }
```

Snippet 5. PrivatePublicSale.sol Redundant statements in `withdraw` function

UPDATES

- *Mar 31, 2022*: This issue has been acknowledged and fixed by the House of Kibaa team.

2.3.4. LandNFT.sol - Redundant state variables in the contract INFORMATIVE

There are some state variables that are unused in the contract but the name of them may cause users confuse. We will list them in the below code:

```
41  uint256 public saleStartsAt;
42      uint256 public publicsaleStartsAt;
```

```
43      uint256 public publicsaleEndsAt;
44      uint256 public privatesaleStartsAt;
45      uint256 public privatesaleEndsAt;
46      uint256 public redemptionEndsAt;
47      uint256 public constant MAX_PRIVATE_SALE_SUPPLY = 3000;
48      uint256 public constant MAX_TOKENS = 10000; // Max number of toke…
    n sold in  sale
```

Snippet 6. LandNFT.sol Redundant state variables

RECOMMENDATION

We suggest removing these variables for readability.

UPDATES

- *Mar 31, 2022*: This issue has been acknowledged and fixed by the House of Kibaa team.

2.3.5. PocketDimensionUser.sol - Using calldata instead of memory for gas saving INFORMATIVE

There are some functions which are using memory. The contract may change to usecalldata for memory.

The list prototype of function which may change:

```
function whitelistMembers (address[] memory users, uint8[] memory balance…
  s) external onlyOwner
function removeWhitelistMembers (address[] memory users) external onlyOwn…
  er
```

Snippet 7. PrivatePublicSale.sol Using calldata instead of memory for gas saving

UPDATES

- *Apr 22, 2022*: This issue has been acknowledged by the House of Kibaa team.

2.3.6. PocketDimensionUser.sol - Redundant require statement in mintInternal function INFORMATIVE

```
function mintInternal(uint8 landType, uint256 numTokens)
        external
    {

        uint256 time = (block.timestamp);
```

```
        require(time < privatesaleStartsAt, "Can only mint before private…
   sale");
        require(msg.sender == hokMinterAddress, "Not allowed");

        require(landType > 0 && landType < 11, "Invalid Land type");

        require(_landTypeCounter[landType] + numTokens < MAX_PRIVATE_LAND…
   _SALE_PER_TYPE + 1, "Sold out");

        require((_tokenIds.current() + numTokens) <= MAX_INTERNAL_SUPPLY,…
   "Maximum number reached");

        for(uint256 i = 0; i < numTokens; i++){
            _tokens[_tokenIds.current()] = landType;
            _tokenIds.increment();
        }
        _landTypeCounter[landType] += numTokens;
        _safeMint(msg.sender, numTokens);
        emit TokenMinted(msg.sender, landType, numTokens);
    }
```

Snippet 8. PrivatePublicSale.sol Redundant require statement in `mintInternal` function

The mintInternal function has been only used before the privatesaleStartsAt time. Where the _landTypeCounter[landType] value is always lower than the MAX_INTERNAL_SUPPLY value (235).And the numTokens is also lower than MAX_INTERNAL_SUPPLY. So the expression _landTypeCounter[landType] + numTokens < MAX_PRIVATE_LAND_SALE_PER_TYPE(#1000) is always true. So the marked require statement is unnecessary.

UPDATES

- *Apr 22, 2022*: This issue has been acknowledged by the House of Kibaa team.

2.3.7. PocketDimensionUser.sol - Missing check length of the array parameter in whitelistMembers function INFORMATIVE

```
function whitelistMembers (address[] memory users, uint8[] memory balance…
  s) external onlyOwner {
        require(!saleActive, "Cannot whitelist");
        uint256 time = (block.timestamp);
        require(time < saleStartsAt, "Cannot modify after private sale st…
  art");
```

```
        for (uint i = 0; i < users.length; i++) {
            _memberslist[users[i]].exists = true;
            _memberslist[users[i]].balance = balances[i];
            _memberslist[users[i]].minted = 0;
        }
    }
```

Snippet 9. PrivatePublicSale.sol Missing check length of the array parameter in `whitelistMembers` function

The function uses the balances array without checking its length. If the length of users is larger than balances length, it will cause an error because of accessing an array out of bound.

RECOMMENDATION

We suggest adding a require statement that checks the length of the balances parameter.

UPDATES

- *Apr 22, 2022*: This issue has been acknowledged by the House of Kibaa team.

2.3.8. PocketDimensionUser.sol - Missing emit event in some functions INFORMATIVE

The contract has some functions which transfer tokens and set important values, but they do not emit an event to notice.

For instance, with withdrawTokens function, the owner transfers the tokens out of the contract without emitting them.

```
function withdrawTokens(address _tokenContract) external onlyOwner nonRee…
  ntrant {
        IERC20 tokenContract = IERC20(_tokenContract);

        // transfer the token from address of hok address
        uint256 _amount = tokenContract.balanceOf(address(this));

        tokenContract.transfer(hokWithdrawAddress, _amount);
    }
```

Snippet 10. PrivatePublicSale.sol Missing emit event in `withdrawTokens` function

The royalty is an important state variable. It should be emitted event when changing.

```
function setRoyalty(uint16 _royalty) external virtual onlyOwner {
        require(_royalty >= 0 && _royalty <= 1000, 'Royalty must be betwe…
  en 0% and 10%.');
```

```
        royalty = _royalty;
    }
```

Snippet 11. PrivatePublicSale.sol Missing emit event in `setRoyalty` function

UPDATES

- *Apr 22, 2022*: This issue has been acknowledged by the House of Kibaa team.

2.3.9. PocketDimensionUser.sol - Best-practice in changeSaleStatus function INFORMATIVE

```
function changeSaleStatus() external onlyOwner{
        // require(msg.sender == saleContractAdddress, "Not Allowed");
        saleActive = !saleActive;
        emit SaleStatusChange(msg.sender, saleActive);
    }
```

Snippet 12. PrivatePublicSale.sol The `changeSaleStatus` function

The changeSaleStatus is used to toggle the saleActive state variable.

With current logic, the owner could not exactly control the next value, if the owner didn't check the previous saleActive value. With a mistake, the transaction is sent twice, the value will be idempotent.

The changeSaleStatus will be more convenient if the owner passes the exact value.

UPDATES

- *Apr 22, 2022*: This issue has been acknowledged by the House of Kibaa team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Mar 18, 2022*	Private Report	Verichains Lab
1.1	*Mar 31, 2022*	Public Report	Verichains Lab
1.2	*Apr 22, 2022*	Public Report	Verichains Lab

Table 2. Report versions history