verichains

SECURITY AUDIT OF

RUNGEM, NFTV2 AND GAME SMART CONTRACTS

Public Report

Oct 28, 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.
BSC	Binance Smart Chain or BSC is an innovative solution for introducing interoperability and programmability on Binance Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Oct 28, 2022. We would like to thank the Runnow 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 RUNGEM, NFTv2 and Game 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 application, along with some recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About RUNGEM, NFTv2 and Game Smart Contracts

Runnow.io is a Lifestyle Gamification - Move & Earn project developed by KBG Studio. They aim to build a healthier world by encouraging people to take steps in daily exercise. By joining multiple sports and competitions, users can improve their physical health, earn valuable in-game rewards, connect with communities worldwide, and give support to those in need.

Runnow.io is developed on the BSC Network which is faster, and more secure with cheaper transaction costs. In the near future, they will support multichain. In addition, the cooperation and investment from GemUni and this is a solution for Runnow.io to inherit the strength of the ecosystem that GemUni has built with many useful supporting features such as NFT for leasing - borrowing, Staking & Farming, Social, etc.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of RUNGEM, NFTv2 and Game Smart Contracts.

It was conducted on commit `8ecc2c0a11f92efd0a7918744b38d78c307dfb45` from git repository link: *https://github.com/RUNNOW-PROJECT/blockchain-contracts*.

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

SHA256 Sum	File
`ed4bc4fdb8d122bddc86469c050b572e2c1dcf4161d0e54239401579c69acd34`	`./game/Game.sol`
`5c130b9444055d880a4858b752a0f1a0b45b478d9d51625943262e352b863b1a`	`./coin/RUNGEM.sol`
`48523394c42c5a207ca6b7009781eb86d5da47a72f2e9c418a7d30adda809c93`	`./nft/RunnowNFTV2.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

The RUNGEM, NFTv2 and Game Smart Contracts was written in `Solidity` language, with the required version to be `^0.8.0`. The source code was written based on OpenZeppelin's library.

2.1.1. RUNGEM contract

PROPERTY	VALUE
Name	RUNGEM
Symbol	RUNGEM
Decimals	18

Table 2. The RUNGEM, NFTv2 and Game Smart Contracts properties

RUNGEM contract extends `ERC20Upgradeable`, `OwnableUpgradeable` and `PausableUpgradeable` abstract contracts. With `OwnableUpgradeable`, by default, Token Owner is the contract deployer but he can transfer ownership to another address at any time.

The contract implements `mint` external function with the `onlyOwner` modifier. So the only owner of the contract can `mint` new tokens which can change the `totalSupply` of the contract.

Note: Sine the RUNGEM contract is upgradable, the contract `owner` can upgrade it with the logic that is not in our audit scope.

2.1.2. RunnowNFTV2 contract

RunnowNFTV2 contract extends `ERC721Upgradeable`, `OwnableUpgradeable` and `EIP712Upgradeable` abstract contracts. With `OwnableUpgradeable`, by default, Token Owner is the contract deployer but he can transfer ownership to another address at any time. The contract also implements a bunch of `pause` functions and modifiers working like the `Pausable` contract. The `owner` can `pause/unpause` contract activities through the function is regarded above.

The contract allows users to `mint` tokens with an amount of ERC20 tokens set by the server through the signature.

The contract also implements `mintFromGame`, `mintFromGameAndBringToGame` and `mintBatch` functions which allow some specific roles in the contract to create new tokens for other users.

Note: Sine the RunnowNFTV2 contract is upgradeable, the contract `owner` can upgrade it with the logic that is not in our audit scope.

2.1.3. Game contract

Game contract extends `ReentrancyGuardUpgradeable`, `OwnableUpgradeable` and `EIP712Upgradeable` abstract contracts. With `OwnableUpgradeable`, by default, Token Owner is the contract deployer but he can transfer ownership to another address at any time. The contract also implements a bunch of `pause` functions and modifiers working like the `Pausable` contract. The `owner` can `pause/unpause` contract activities through the function that is regarded above.

The contract allows users to deposit their assets to the game freely. When users want to withdraw their assets, they must be approved by the server through signature.

Note: Sine the Game contract is upgradeable, the contract `owner` can upgrade it with the logic that is not in our audit scope.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of RUNGEM, NFTv2 and Game Smart Contracts.

Runnow fixed the code according to Verichain's draft report in commit `a2b81355644a61fbf69bfea7a79935a35afaa395`.

2.2.1. RunnowNFTV2.sol - Attacker can front running attack to steal voucher MEDIUM

In the `redeem` function, the `data.signature` doesn't cover the user address. Therefore, the attacker can listen to the mempool and observe all transactions which are sent to this contract with `redeem` signature. When the attacker finds a voucher with a valuable item, he can send a `redeem` transaction including that voucher with the higher `gasprice` (higher gas price transaction is usually mined first) than the user's one. The result is the user transaction is reverted and the voucher is used for the attacker.

```
function redeem(ItemVoucherStruct calldata data) public payable {
        // Make sure signature is valid and get the address of the signer
        address signer = _verifyItemVoucher(data); //<-- The mistake statement
        // Make sure that the signer is authorized to mint an item
        require(signer == owner(), "Signature invalid or unauthorized");

        // Check nonce
        require(!_noncesMap[data.nonce], "The nonce has been used");
        _noncesMap[data.nonce] = true;
        ...

    function _verifyItemVoucher(ItemVoucherStruct calldata data)
        internal
        view
        returns (address)
    {
```

```
        bytes32 digest = _hashItemVoucher(data);
        return ECDSAUpgradeable.recover(digest, data.signature); //<-- The signature
doesn't cover user address
    }

    function _hashItemVoucher(ItemVoucherStruct calldata data)
        internal
        view
        returns (bytes32)
    {
    return
        _hashTypedDataV4(
            keccak256(
            abi.encode(
                keccak256(
                    "ItemVoucherStruct(string id,string itemType,string extraType,uint256
price,address tokenAddress,string nonce)"
                ),
                keccak256(bytes(data.id)),
                keccak256(bytes(data.itemType)),
                keccak256(bytes(data.extraType)),
                data.price,
                data.tokenAddress,
                keccak256(bytes(data.nonce))
            )
        )
        );
    }
```

UPDATES

- *Oct 27, 2022*: This issue has been acknowledged and fixed by the Runnow team.

2.2.2. Upgradeable contract MEDIUM

The contracts in audit scope inherit `upgradeable` contracts which allow the deployer to change the logic. Any compromise to the `deployer` account may allow the hacker to take advantage of this.

UPDATES

- *Oct 27, 2022*: This issue has been acknowledged by the Runnow team.

2.2.3. RunnowNFTV2.sol - Missing token transfer result checking in `redeem` function INFORMATIVE

In the `redeem` function of the `RunnowNFT` contract, the transfer result is not checked. So, if the `transferFrom` method of the `data.tokenAddress` does not revert when the transferring is failed, the contract using this function may not work properly.

```
function redeem(ItemVoucherStruct calldata data) public payable {
    ...
    // Transfer payment
    if(data.tokenAddress == address(0)){
        require(msg.value >= data.price, "Not enough money");
        (bool success, ) = devWalletAddress.call{value: msg.value}("");
        require(success, "Transfer payment failed");
    }
    else{
        IERC20Upgradeable(data.tokenAddress).transferFrom(msg.sender, devWalletAddress,
data.price);
    }
    ...
}
```

RECOMMENDATION

We should use the `safeTransfer` function from `SafeERC20` library when transferring token.

UPDATES

- *Oct 27, 2022*: This issue has been acknowledged and fixed by the Runnow team.

2.2.4. Centralized mechanisms INFORMATIVE

Currently, there are some centralized logic flows included in the contracts.

First - the `withdraw` tokens/items logic in the `game` contract, the users `withdraw` tokens/items through `data.signature` which is signed by the team's server. There is no constraint between the input and output tokens/items. If an attacker controls the server, he can create any withdraw signature to withdraw all things in the `game` contract.

Second, both `RUNGEM` and `RunnowNFT` contracts include the `mint` function, any compromise to the `owner` account may allow the hacker to create a large number of tokens to destroy the liquid pool.

RECOMMENDATION

We strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices.

UPDATES

- *Oct 27, 2022*: This issue has been acknowledged and fixed by the Runnow team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 27, 2022*	Public Report	Verichains Lab
1.0	*Oct 28, 2022*	Public Report	Verichains Lab

Table 3. Report versions history