verichains

SECURITY AUDIT OF

RUNNOW SMART CONTRACTS

RUNNOW

MOVE & EARN

Public Report

August 10, 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 prepared by Verichains Lab on August 10, 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 Runnow 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. Runnow team has resolved and updated all issues following our recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY **5**

1.1. About Runnow Smart Contracts **5**

1.2. Audit scope **5**

1.3. Audit methodology **6**

1.4. Disclaimer **7**

2. AUDIT RESULT **8**

2.1. Overview **8**

2.1.1. Runnow vesting contract 8

2.1.2. Marketplace contract 9

2.2. Findings **9**

2.2.1. Marketplace.sol - Missing NFT and offer ownership check when updating offer CRITICAL 10

2.2.2. Marketplace.sol - Front-running in buy function CRITICAL 11

2.2.3. MarketplaceV2.sol - User can buy listed NFTs without money CRITICAL 12

2.2.4. RunnowVesting.sol - Missing a month when distributing for farming and staking CRITICAL 15

2.2.5. RunnowVesting.sol - Public sale amount is not distributed for market and community CRITICAL 15

2.2.6. RunnowVesting.sol - Missing a month when distributing for market and community CRITICAL 16

2.2.7. Marketplace.sol - The remaining money is stuck inside the contract LOW 17

2.2.8. Marketplace.sol - Reentrancy attack to create fake offers LOW 18

2.3. Additional notes and recommendations **19**

2.3.1. RunnowVesting.sol - Redundant condition in `getAmountForLiquidity` function INFORMATIVE 19

3. VERSION HISTORY **21**

1. MANAGEMENT SUMMARY

1.1. About Runnow 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 Runnow Smart Contracts.

It was conducted on commit `cd5b7aaa5970b7515ff05a58585aeff44d08ef10` 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
`87d0124d8ff4bc15b4c43d7e229a04c4efd5c3019e8b393e2362a93afab41451`	`./vesting/RunnowVesting.sol`
`01f6c194d1b290ba6453cbffd16c7dd3aed6fcb40ddbd307dc49de17f24cf992`	`./vesting/IRunnow.sol`
`8ceb77bcdbe3c056294ba47a46e1235f454547c38bf5120243ccede7537a99d6`	`./marketplace/Marketplace.sol`
`e4cc6def9235d37718948b78f89554e66a38b3b0fa26503c75cc43fe0c312433`	`./marketplace/MarketplaceV2.sol`
`05b333e91200491bf38a972178e662200c1feb5746820c7b04986b2ecc82a367`	`./marketplace/MarketplaceV3.sol`

Notes:

- *August 02, 2022*: The `Marketplace.sol` file is updated, and the new source is put inside the `MarketplaceV2.sol` file.
- *August 10, 2022*: The `MarketplaceV2.sol` file is updated, and the new source is put inside the `MarketplaceV3.sol` file.

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 Runnow 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. Runnow vesting contract

The logic for the token vesting contract is defined in `RunnowVesting.sol`, this contract implements a vesting mechanism to lock and release tokens according to a configured schedule defined by the contract owner. The token release schedules can be summarized as below:

	%	Token Amount	% Value	Release Schedule
Seed Sales	3%	30,000,000	$300,000	TGE 0%, Cliff for 3 months then linear monthly vesting in 12 months
Private Sales	4%	40,000,000	$480,000	TGE 0%, Cliff for 3 months then linear monthly vesting in 12 months
Public Sales (IDO)	6%	60,000,000	$840,000	TGE 20%, 1 month cliff then linear vesting in 3 months
Advisors & Partners	5%	50,000,000	$700,000	12 months Cliff, then linear vesting in 24 months
Team	20%	200,000,000	$2,800,000	12 months Cliff, then linear vesting in 24 months
Marketing & Operation	10%	100,000,000	$1,400,000	TGE 1% then linear vesting in 36 months from TGE
Rewards Treasury	35%	350,000,000	$4,900,000	1 month Cliff then linear vesting in 36 months from TGE
Farming & Staking	15%	150,000,000	$2,100,000	1 month Cliff then linear vesting in 36 months from TGE

	%	Token Amount	% Value	Release Schedule
Liquidity & Listing	2%	20,000,000	$280,000	TGE 5%, monthly vesting in 6 months
Total	100%	1,000,000,000		

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

2.1.2. Marketplace contract

The marketplace is where Runnow.io delivers NFT Boxes to the market with a limited number of versions perversion.

Where Users can sell and buy, auction, NFTs, etc. together to be able to use in Runnow.io.

To Sell NFT, the user must transfer the NFT out of the application to the marketplace.

To load the NFT into Runnow.io, the user must buy, rent / borrow it from the owner.

The marketplace contract extends `OwnableUpgradeable`, `EIP712Upgradeable` and `ReentrancyGuardUpgradeable` contracts. With `OwnableUpgradeable`, the contract owner is also contract deployer, he can set the marketplace fees and transfer ownership to another address at any time.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Runnow Smart Contracts. Runnow team fixed the code, according to Verichains's draft report.

#	Issue	Severity	Status
1	Missing NFT and offer ownership check when updating offer	CRITICAL	Fixed
2	Front-running in buy function	CRITICAL	Fixed
3	User can buy listed NFTs without money	CRITICAL	Fixed
4	Missing a month when distributing for farming and staking	CRITICAL	Fixed
5	Public sale amount is not distributed for market and community	CRITICAL	Fixed

#	Issue	Severity	Status
6	Missing a month when distributing for market and community	CRITICAL	Fixed
7	The remaining money is stuck inside the contract	LOW	Fixed
8	Reentrancy attack to create fake offers	LOW	Fixed

Audit team also suggested some possible enhancements.

#	Issue	Status
1	Redundant condition in `getAmountForLiquidity` function	Fixed

2.2.1. Marketplace.sol - Missing NFT and offer ownership check when updating offer CRITICAL

The `offer` function allows users to update the previous offer by using the existing `data.id`. However, when updating an offer, users can easily input malicious values for `itemAddress`, `tokenId`, or `price`. A simple attack scenario is that a user can update his offer point to the existing NFT item he doesn't own. By setting the price to zero or a low value, they can easily buy and take over it.

```
function offer(OrderItemStruct calldata data) public {
    // Make sure signature is valid and get the address of the signer
    address signer = _verifyOrderItem(data);
    // Make sure that the signer is authorized to offer item
    require(signer == owner(), "Signature invalid or unauthorized");

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

    if (!itemsMap[data.id].isExist) {
        IERC721Upgradeable(data.itemAddress).transferFrom(
            _msgSender(),
            address(this),
            data.tokenId
        );
    }
    // MISSING OFFER UPDATE CHECK

    itemsMap[data.id] = ItemStruct({
        id: data.id,
        itemType: data.itemType,
        extraType: data.extraType,
```

```
        itemAddress: data.itemAddress,
        tokenId: data.tokenId,
        owner: _msgSender(),
        price: data.price,
        isExist: true
    });
    // ...
}
```

RECOMMENDATION

The update offer function should be implemented separately. Moreover, fields that can be updated should also be restricted (E.g. `itemAddress` and `tokenId` should not be allowed to update).

UPDATES

- *Aug 02, 2022*: This issue has been acknowledged and fixed by the Runnow team. They've added ownership checking before updating the offer. The `itemAddress` and `tokenId` are verified and signed by the marketplace contract owner to ensure accuracy before updating.

2.2.2. Marketplace.sol - Front-running in buy function CRITICAL

The `buy` function in the `Marketplace` contract is vulnerable to front-running attack. The attacker can listen for pending `buy(id)` transaction from buyers and quickly update the offer price for that `id` (the offer signature should be prepared beforehand). This causes user to buy the NFT item with an unexpected price.

```
function buy(string memory id) public payable nonReentrant {
    // Check exists & don't buy own
    require(itemsMap[id].isExist, "Item is not in marketplace");
    require(
        itemsMap[id].owner != _msgSender(),
        "You cannot buy your own item"
    );

    ItemStruct memory item = itemsMap[id];

    // Transfer payment
    require(msg.value >= item.price, "Not enough money");
    // ...
}
```

RECOMMENDATION

A `maxPrice` parameter should be added to the `buy` function as below:

```
function buy(string memory id, uint256 maxPrice) public payable nonReentrant {
    // Check exists & don't buy own
    require(itemsMap[id].isExist, "Item is not in marketplace");
    require(
        itemsMap[id].owner != _msgSender(),
        "You cannot buy your own item"
    );

    ItemStruct memory item = itemsMap[id];

    // Transfer payment
    require(item.price <= maxPrice, "Invalid price"); // CHECK maxPrice
    require(msg.value >= item.price, "Not enough money");
    // ...
}
```

UPDATES

- *Aug 02, 2022*: This issue has been acknowledged and fixed by the Runnow team by checking the amount buyer willing to send with the `tokenPrice`. However, while fixing this bug, they introduced a new bug in the `MarketplaceV2` contract.

2.2.3. MarketplaceV2.sol - User can buy listed NFTs without money CRITICAL

There are two critical bugs in the `buy` function in the `MarketplaceV2` contract:

The first one, if a payment `tokenAddress` is not supported by the token owner via the `offer` function call, the value of `tokenPrice[id][tokenAddress]` would be `0`. So, any user can buy that NFT item at a zero price.

The second one, consider the case when a user wants to buy a listed NFT item with the native coin. However, instead of sending the correct value for `msg.value`, he can set the amount equal to that value. Finally, he can pass the price check line below and buy the NFT item successfully.

```
require(
    msg.value == tokenPrice[id][tokenAddress] || tokenPrice[id][tokenAddress] == amount,
    "Not enough money"
);
function buy(string memory id, address tokenAddress,uint256 amount)
    public
    payable
    nonReentrant
{
    // ...
    require(allowedToken[tokenAddress], "Token not for sale");
    require(
        msg.value == tokenPrice[id][tokenAddress] || tokenPrice[id][tokenAddress] ==
amount,
```

```
        "Not enough money"
    ); // ERROR

    ItemStruct memory item = itemsMap[id];

    uint256 totalFeesShareAmount = (tokenPrice[id][tokenAddress] *
        feesCollectorCutPerMillion) / 1_000_000;
    uint256 ownerShareAmount = tokenPrice[id][tokenAddress] -
        totalFeesShareAmount;

    // Transfer payment
    if (tokenAddress == address(0)) {
        //transfer with BNB
        if (totalFeesShareAmount > 0) {
            (bool success, ) = feesCollectorAddress.call{
                value: totalFeesShareAmount
            }("");
            require(success, "Transfer fee failed");
        }

        (bool success, ) = item.owner.call{value: ownerShareAmount}("");
        require(success, "Transfer money failed");
    } else {
        // transfer with token
        IERC20Upgradeable(tokenAddress).transferFrom(
            _msgSender(),
            feesCollectorAddress,
            totalFeesShareAmount
        );
        IERC20Upgradeable(tokenAddress).transferFrom(
            _msgSender(),
            item.owner,
            ownerShareAmount
        );
    }
    // ...
}
```

RECOMMENDATION

Don't use `tokenPrice` mapping for price checking, use the `priceList` field in the Offer item instead.

UPDATES

- *Aug 02, 2022*: This issue has been acknowledged and fixed by the Runnow team. For the first issue, they've updated the contract to check the `tokenAddress` if it is included in the list offered by the NFT's owner. For the second one, they've added the requirement

before transferring the payment to ensure `msg.value == amount == tokenPrice`. It is to prevent front-running attack and manipulation between `msg.value` and `amount`.

```
function buy(string memory id, address tokenAddress,uint256 amount)
    public
    payable
    nonReentrant
{
    // Check exists & don't buy own
    require(itemsMap[id].isExist, "Item is not in marketplace");
    require(
        itemsMap[id].owner != _msgSender(),
        "You cannot buy your own item"
    );
    require(allowedToken[tokenAddress], "Token not for sale");
    require(tokenPrice[id][tokenAddress] > 0, "Token not listed");

    ItemStruct memory item = itemsMap[id];

    uint256 totalFeesShareAmount = (tokenPrice[id][tokenAddress] *
        feesCollectorCutPerMillion) / 1_000_000;
    uint256 ownerShareAmount = tokenPrice[id][tokenAddress] -
        totalFeesShareAmount;

    // Transfer payment
    if (tokenAddress == address(0)) {
        require(msg.value == amount && msg.value == tokenPrice[id][tokenAddress],"Not same
price");
        //transfer with BNB
        if (totalFeesShareAmount > 0) {
            (bool success, ) = feesCollectorAddress.call{
                value: totalFeesShareAmount
            }("");
            require(success, "Transfer fee failed");
        }

        (bool success, ) = item.owner.call{value: ownerShareAmount}("");
        require(success, "Transfer money failed");
    } else {
        require(amount == tokenPrice[id][tokenAddress],"Not same price");
        // transfer with token
        IERC20Upgradeable(tokenAddress).transferFrom(
            _msgSender(),
            feesCollectorAddress,
            totalFeesShareAmount
        );
        IERC20Upgradeable(tokenAddress).transferFrom(
            _msgSender(),
            item.owner,
            ownerShareAmount
```

```
        );
    }
    // ...
}
```

2.2.4. RunnowVesting.sol - Missing a month when distributing for farming and staking CRITICAL

In the `getAmountForFarmingAndStaking` function, the `linearAmount` is distributed for months from `1` to `35`. So, one distribution month is missing (35/36).

```
function getAmountForFarmingAndStaking(uint256 month)
    public
    view
    returns (uint256 amount)
{
    uint256 maxAmount = 50_000_000 * 10**decimals;
    uint256 linearAmount = maxAmount / 36;
    if (month > 36 || month == 0) amount = 0;
    else if (month >= 0 && month < 35) amount = linearAmount;
    else if (month == 35) amount = maxAmount - linearAmount * 35;
}
```

UPDATES

- *Aug 02, 2022*: This issue has been acknowledged and fixed by the Runnow team.

```
function getAmountForFarmingAndStaking(uint256 month)
    public
    view
    returns (uint256 amount)
{
    uint256 maxAmount = 150_000_000 * 10**decimals;
    uint256 linearAmount = maxAmount / 36;
    if (month > 36 || month == 0) amount = 0;
    else if (month > 0 && month <= 35) amount = linearAmount;
    else if (month == 36) amount = maxAmount - linearAmount * 35;
}
```

2.2.5. RunnowVesting.sol - Public sale amount is not distributed for market and community CRITICAL

In the `getAmountForMktAndCommunity` function, the `publicSaleAmount` is extracted from the `maxAmount`. However, it is not being distributed to the market and community.

```
function getAmountForMktAndCommunity(uint256 month)
    public
    view
    returns (uint256 amount)
{
```

```
    uint256 maxAmount = 150_000_000 * 10**decimals;
    uint256 publicSaleAmount = 1_500_000 * 10**decimals; // NOT BEING DISTRIBUTED
    uint256 linearAmount = (maxAmount - publicSaleAmount) / 36;
    if (month > 36) amount = 0;
    else if (month >= 0 && month < 35) amount = linearAmount;
    else if (month == 36) amount = maxAmount - linearAmount * 35;
}
```

UPDATES

- *Aug 02, 2022*: This issue has been acknowledged and fixed by the Runnow team.

```
function getAmountForMktAndCommunity(uint256 month)
    public
    view
    returns (uint256 amount)
{
    uint256 maxAmount = 100_000_000 * 10**decimals;
    uint256 publicSaleAmount = 1_000_000 * 10**decimals;
    uint256 linearAmount = (maxAmount - publicSaleAmount) / 36;
    if (month > 36) amount = 0;
    else if (month == 0 ) amount = publicSaleAmount;
    else if (month >= 1 && month <= 35) amount = linearAmount;
    else if (month == 36) amount = maxAmount - linearAmount * 35;
}
```

2.2.6. RunnowVesting.sol - Missing a month when distributing for market and community CRITICAL

In the `getAmountForMktAndCommunity` function, the `linearAmount` should be distributed for months from 1 to 36. However, month `35` is missing.

```
function getAmountForMktAndCommunity(uint256 month)
    public
    view
    returns (uint256 amount)
{
    uint256 maxAmount = 100_000_000 * 10**decimals;
    uint256 publicSaleAmount = 1_000_000 * 10**decimals;
    uint256 linearAmount = (maxAmount - publicSaleAmount) / 36;
    if (month > 36) amount = 0;
    else if (month == 0 ) amount = publicSaleAmount;
    else if (month >= 1 && month < 35) amount = linearAmount;
    else if (month == 36) amount = maxAmount - linearAmount * 35;
}
```

UPDATES

- *Aug 02, 2022*: This issue has been acknowledged and fixed by the Runnow team by updating the distribution for month 35.

```
function getAmountForMktAndCommunity(uint256 month)
    public
    view
    returns (uint256 amount)
{
    uint256 maxAmount = 100_000_000 * 10**decimals;
    uint256 publicSaleAmount = 1_000_000 * 10**decimals;
    uint256 linearAmount = (maxAmount - publicSaleAmount) / 36;
    if (month > 36) amount = 0;
    else if (month == 0 ) amount = publicSaleAmount;
    else if (month >= 1 && month <= 35) amount = linearAmount;
    else if (month == 36) amount = maxAmount - linearAmount * 35;
}
```

2.2.7. Marketplace.sol - The remaining money is stuck inside the contract LOW

In the `buy` function, if user send more money than the item price, the remaining money would be hold inside the contract without any mechanism to get it back.

```
function buy(string memory id) public payable nonReentrant {
    // ...

    // Transfer payment
    require(msg.value >= item.price, "Not enough money");

    uint256 totalFeesShareAmount = (item.price *
        feesCollectorCutPerMillion) / 1_000_000;

    if (totalFeesShareAmount > 0) {
        (bool success, ) = feesCollectorAddress.call{value:totalFeesShareAmount}("");
        require(success, "Transfer fee failed");
    }

    uint256 ownerShareAmount = item.price - totalFeesShareAmount; // REMAINING MONEY IS
STUCK
    (bool success, ) = item.owner.call{value: ownerShareAmount}("");
    require(success, "Transfer money failed");
}
```

RECOMMENDATION

The remaining amount can be sent to the offer owner (the seller) or the fee collector address.

UPDATES

- *Aug 02, 2022*: This issue has been acknowledged and fixed by the Runnow team. The contract is updated to ensure that the buyer sends the correct amount (cannot send more than the item price).

2.2.8. Marketplace.sol - Reentrancy attack to create fake offers LOW

The `buy` function in the `Marketplace` contract is also vulnerable to front-running attacks. The NFT owner can be a contract that contains reentrancy attack code inside the fallback function, so that when the money is transferred to the owner, the following actions can happen inside the attack contract:

- Callback to the `Marketplace` contract to withdraw the buying NFT item.
- Re-offer that item to the marketplace (with a new offer id).

After these actions, the NFT owner can create a new offer without the corresponding NFT item inside the `Marketplace` contract (that NFT item has been transferred by the `buy` transaction before).

```
function buy(string memory id) public payable nonReentrant {
    // Check exists & don't buy own
    require(itemsMap[id].isExist, "Item is not in marketplace");
    require(
        itemsMap[id].owner != _msgSender(),
        "You cannot buy your own item"
    );

    ItemStruct memory item = itemsMap[id];

    // Transfer payment
    require(msg.value >= item.price, "Not enough money");

    uint256 totalFeesShareAmount = (item.price *
        feesCollectorCutPerMillion) / 1_000_000;

    if (totalFeesShareAmount > 0) {
        (bool success, ) = feesCollectorAddress.call{value:totalFeesShareAmount}("");
        require(success, "Transfer fee failed");
    }

    uint256 ownerShareAmount = item.price - totalFeesShareAmount;
    (bool success, ) = item.owner.call{value: ownerShareAmount}(""); // FRONT-RUNNING
ATTACK
    require(success, "Transfer money failed");

    IERC721Upgradeable(item.itemAddress).transferFrom(
        address(this),
        _msgSender(),
        item.tokenId
    );
    // ...
}
```

RECOMMENDATION

The `nonReentrant` modifier must be added to all functions that can be used for reentrancy attacks (E.g. `withdraw`, `buy`, `offer`).

UPDATES

- *Aug 02, 2022*: This issue has been acknowledged and fixed by the Runnow team. The `nonReentrant` modifier has been added to `withdraw`, `buy`, and `offer` functions to avoid reentrancy attacks from external contracts.

2.3. Additional notes and recommendations

2.3.1. RunnowVesting.sol - Redundant condition in `getAmountForLiquidity` function INFORMATIVE

There is a redundant condition `if ((month >= 13) || month > 15)` inside the `getAmountForLiquidity` function.

```
function getAmountForLiquidity(uint256 month)
    public
    view
    returns (uint256 amount)
{
    uint256 maxAmount = 50_000_000 * 10**decimals;
    uint256 publicSaleAmount = 2_500_000 * 10**decimals;
    uint256 linearAmount = (maxAmount - publicSaleAmount) / 12;
    if (month == 0) amount = publicSaleAmount;
    else if ((month >= 13) || month > 15) amount = 0; // REDUNDANT
    else if (month >= 1 && month <= 11) amount = linearAmount;
    else if (month == 12)
        amount = maxAmount - publicSaleAmount - linearAmount * 11;
}
```

RECOMMENDATION

The above function can be fixed as below:

```
function getAmountForLiquidity(uint256 month)
    public
    view
    returns (uint256 amount)
{
    uint256 maxAmount = 50_000_000 * 10**decimals;
    uint256 publicSaleAmount = 2_500_000 * 10**decimals;
    uint256 linearAmount = (maxAmount - publicSaleAmount) / 12;
    if (month == 0) amount = publicSaleAmount;
    else if (month >= 13) amount = 0; // FIXED
```


```
    else if (month >= 1 && month <= 11) amount = linearAmount;
    else if (month == 12)
        amount = maxAmount - publicSaleAmount - linearAmount * 11;
}
```

UPDATES

- *Aug 02, 2022*: This issue has been acknowledged and fixed by the Runnow team by removing the redundant condition `month > 15`.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Aug 02, 2022*	Public Report	Verichains Lab
1.1	*Aug 10, 2022*	Public Report	Verichains Lab

Table 2. Report versions history