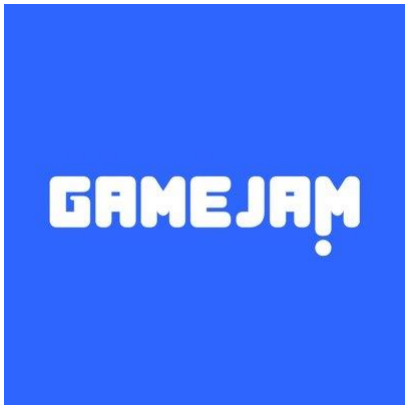
verichains

SECURITY AUDIT OF

GAMEJAM AIRDROP SMART CONTRACTS



Public Report

Nov 09, 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 was prepared by Verichains Lab on Nov 09, 2022. We would like to thank the Gamejam 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 Gamejam AirDrop 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 no vulnerable issue in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY **5**

1.1. About Gamejam AirDrop Smart Contracts **5**

1.2. Audit scope **5**

1.3. Audit methodology **5**

1.4. Disclaimer **6**

2. AUDIT RESULT **7**

2.1. Overview **7**

2.1.1. JamPlatformToup contract 7

2.1.2. PlaylinkAirDrop contract 7

2.2. Findings **7**

2.3. Additional notes and recommendations **7**

2.3.1. PlaylinkAirdrop.sol - `Operator` may re airdrop `asset.availableAmount` equals `0` INFORMATIVE 7

2.3.2. JamPlatformTopUp.sol - Consider using EnumerableMap instead of using nested for-loops in `whitelistCurrencies` function for gas-saving INFORMATIVE 8

2.3.3. PlaylinkAidrop.sol - Centralized mechanism INFORMATIVE 9

2.3.4. PlaylinkAirdrop.sol - Mistake in enum value INFORMATIVE 10

2.3.5. PlaylinkAirdrop.sol - Use `calldata` instead of `memory` for gas saving INFORMATIVE 10

3. VERSION HISTORY **11**

1. MANAGEMENT SUMMARY

1.1. About Gamejam AirDrop Smart Contracts

Gamejam is a decentralized publishing platform and NFT-agnostic games ecosystem on the blockchain.

1.2. Audit scope

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

It was conducted on commit `0ccabafefcaf040bb8a2ea17f015dac0db893a8f` from git repository *https://github.com/gamejamco/jam-contract-core/blob/develop/contracts/*.

There are two files in out audit scope, they are `PlaylinkAirdrop.sol` and `JamPlatformToup.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 Gamejam AirDrop Smart Contracts was written in `Solidity` language, with the required version to be `^0.8.15`.

2.1.1. JamPlatformToup contract

A contract allows users to top up tokens whitelisted by the contract `owner` to the platforms.

2.1.2. PlaylinkAirDrop contract

An airdrop contract that allows users to create airdrop campaigns. Only campaign owners can modify their one. The `operator` can support users to airdrop their campaigns.

2.2. Findings

During the audit process, the audit team found no vulnerability in the given version of Gamejam AirDrop Smart Contracts.

2.3. Additional notes and recommendations

2.3.1. PlaylinkAirdrop.sol - `Operator` may re airdrop `asset.availableAmount` equals `0` INFORMATIVE

The `operator` uses the `airdrop` function to airdrop token for `recipient`s and set `asset.availableAmount` to `0`. The functiondoesn't require the `asset.availableAmount` must be larger than `0`. Therefore, the `operator` may re-airdrop these assets (with `0` value).

With the natural logic, the seller has a feature to trigger `operator` to call airdrop with the seller's data. If there is no limit in this feature, the seller can trigger `operator` several times to airdrop assets with empty value to drain the `operator` balance.

RECOMMENDATION

The code can be fixed as below.

```
function airdrop(
    string memory campaignId,
    uint256[] memory assetIndexes,
    address[] memory recipients
) external onlyOperators nonReentrant {
    ...
    Asset[] memory airdroppedAssets = new Asset[](assetIndexes.length);
    for (uint256 i = 0; i < assetIndexes.length; i++) {
        airdroppedAssets[i] = campaign.assets[assetIndexes[i]];
```

```
        Asset storage asset = campaign.assets[assetIndexes[i]];
        require(asset.availableAmount > 0,"PlaylinkAidrop: re-airdrop is not allowed!")
        if (asset.assetType == AssetType.ERC20) {
            bool success = IERC20(asset.assetAddress).transferFrom(
                campaign.creator,
                recipients[i],
                asset.availableAmount
            );
            require(
                success,
                "PlaylinkAirdrop: failed to send ERC20 assets"
            );
            campaign.totalAvailableAssets -= asset.availableAmount;
            asset.availableAmount = 0;
        } else if (asset.assetType == AssetType.ERC721) {
            IERC721(asset.assetAddress).transferFrom(
                campaign.creator,
                recipients[i],
                asset.assetId
            );
            campaign.totalAvailableAssets--;
            asset.availableAmount = 0;
        } else if (asset.assetType == AssetType.ERC1155) {
            IERC1155(asset.assetAddress).safeTransferFrom(
                campaign.creator,
                recipients[i],
                asset.assetId,
                asset.availableAmount,
                abi.encodePacked("Airdrop ERC1155 assets")
            );
            campaign.totalAvailableAssets -= asset.availableAmount;
            asset.availableAmount = 0;
        }
    }
    ...
}
```

UPDATES

- *Nov 09, 2022*: This issue has been acknowledged and fixed by the Gamejam team in commit `2020600f1ef9bb8f2b9d476d81caace43345f7fb`.

2.3.2. JamPlatformTopUp.sol - Consider using EnumerableMap instead of using nested for-loops in `whitelistCurrencies` function for gas-saving INFORMATIVE

The `whitelistCurrencies` function uses nested for-loops to modify the `_whitelistedCurrencies` state. With the current implementation, the `owner` must pay a high gas value for each function call, the Gamejam Team may change nested for-loops to `EnumerableMap` to reduce the gas cost in transactions.

```
function whitelistCurrencies(
        address[] memory currencies,
        bool[] memory isWhitelisteds
    ) external onlyOwner {
        require(
            currencies.length == isWhitelisteds.length,
            "JamPlatformTopup: lengths mismatch"
        );
        for (uint256 i = 0; i < currencies.length; i++)
            if (isWhitelisteds[i]) {
                if (!_isCurrencyWhitelisted[currencies[i]]) {
                    _isCurrencyWhitelisted[currencies[i]] = true;
                    _whitelistedCurrencies.push(currencies[i]);
                }
            } else {
                if (_isCurrencyWhitelisted[currencies[i]]) {
                    _isCurrencyWhitelisted[currencies[i]] = false;
                    for (uint256 j = 0; j < _whitelistedCurrencies.length; j++)
                        if (_whitelistedCurrencies[j] == currencies[i]) {
                            _whitelistedCurrencies[j] = _whitelistedCurrencies[
                                _whitelistedCurrencies.length - 1
                            ];
                            _whitelistedCurrencies.pop();
                            break;
                        }
                }
            }
    }
```

Snippet 1. The nested for-loops should be changed to EnumerableMap for gas-saving

UPDATES

- *Nov 09, 2022*: This issue has been acknowledged by the Gamejam team.

2.3.3. PlaylinkAidrop.sol - Centralized mechanism INFORMATIVE

Currently, the contract uses a centralized mechanism to allow the `owner` to withdraw `AirdropFee` anytime and control roles in the contract. Any compromise to the `owner` account may allow the hacker to take advantage of this.

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

- *Nov 09, 2022*: This issue has been acknowledged by the Gamejam team.

2.3.4. PlaylinkAirdrop.sol - Mistake in enum value INFORMATIVE

The contract declares `AssetType` as an enum type with `3` values.

```
enum AssetType {
    ERC20,
    ERC721,
    ERC1155
}
```

The range of the `AssetType` when inferred to `uint256` is from `0` to `2`.

But the range require statements in contract are set `3`.

```
require(
    uint256(asset.assetType) <= 3,
    "PlaylinkAirdrop: invalid asset type"
)
```

UPDATES

- *Nov 09, 2022*: This issue has been acknowledged and fixed by the Gamejam team in commit `2020600f1ef9bb8f2b9d476d81caace43345f7fb`.

2.3.5. PlaylinkAirdrop.sol - Use `calldata` instead of `memory` for gas saving INFORMATIVE

In `external` function with array arguments, using `memory` will force solidity to copy that array to memory thus wasting more gas than using directly from calldata. Unless you want to write to the variable, always using `calldata` for external function.

```
function setOperators(address[] memory operators, bool[] memory isOperators) external
onlyOwner {};
function createAirdropCampaign( string memory campaignId, Asset[] memory assets, uint256
startingTime) external payable nonReentrant {};
function updateCampaign( string memory campaignId, Asset[] memory assets, uint256
startingTime) external payable nonReentrant {};
function airdrop( string memory campaignId, uint256[] memory assetIndexes, address[] memory
recipients) external onlyOperators nonReentrant {};
```

RECOMMENDATION

Change `memory` to `calldata` for gas saving in all external function.

UPDATES

- *Nov 09, 2022*: This issue has been acknowledged and fixed by the Gamejam team in commit `2020600f1ef9bb8f2b9d476d81caace43345f7fb`.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Nov 08, 2022*	Private Report	Verichains Lab
1.1	*Nov 09, 2022*	Public Report	Verichains Lab

Table 2. Report versions history