*SECURITY AUDIT OF*

# BLOCKCHAIN MONSTER HUNT SMART CONTRACTS

**Public Report**

*Oct 30, 2021*

## 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 Oct 30, 2021. We would like to thank the Blockchain Monster Hunt 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 Blockchain Monster Hunt 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 Blockchain Monster Hunt Smart Contracts

Blockchain Monster Hunt (BCMH) is the world's first multi-chain game that runs entirely on the blockchain itself. Inspired by Pokémon-GO, BCMH allows players to continuously explore brand new places on the blockchain to hunt and battle monsters. Each block on the blockchain is a unique digital space where a limited number of monsters (of the same DNA gene and rarity) may exist. Players and collectors can hunt or battle for a chance to capture these unique monsters and to earn coins.

## 1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Blockchain Monsters Hunt game. It was conducted on the source code provided by the Blockchain Monster Hunt team.

## 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.

This report contains sensitive information or information that’s not meant to be for the general public. These will be censored out as requested by Blockchain Monster Hunt and will be displayed as [redacted].

# 2. AUDIT RESULT

## 2.1. Overview

The initial review was conducted on Sep 21, 2021 and a total effort of 7 working days was dedicated to identifying and documenting security issues in the code base of the Blockchain Monster Hunt Smart Contracts.

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

| FILE | SHA256 SUM |
|---|---|
| **bcmcerc20.sol** | 0f5ab7e4dbb918c9fcabbf1ffa01318c8f213f594995956006fe21f2196a1d15 |
| **bcmerc721.sol** | 94606edef828417fc6165a7b54537aac374ca66c0049c32d7070b55b9836f1a4 |
| **bcmportalin.sol** | f1935a857145cc0e1988a03d33580301f47cb5f8034a030da9233300673bfc4d |
| **bcmportalinterface.sol** | f501e9a4c1fcfeb6f783460e1914e002a8be28d6dd527dc920a54f3aac236d16 |
| **bcmportalout.sol** | c12713d9f991f36fe280165d14fb21bfc8a6dd50d72d19e2cf0b17b7a0ae1ea0 |
| **bcmsettings.sol** | 19a6e22f3b9f6b6ef572846564210952c1c6255b6a437f75b3e1516f08f6459b |
| **bcmtrade.sol** | 0806d801f25d84da42edca96a3a0e05e4f9241c87113e0595fd52432f1319350 |
| **bcmupgradable.sol** | 27404e32abd5ecf69238eb8eb4753932f3d97148bce4c078c2b84155c7b0b74a |
| **utils.sol** | 9143a9e11774efde5b3519ff24f395d28802de45a7b27c3165af892e4006f6f3 |

## 2.2. Findings

The Blockchain Monster Hunt Smart Contracts was written in Solidity language, with the required version to be ^0.8.2. The source code was written based on OpenZeppelin's library.

Blockchain Monster Hunt fixed the code according to Verichain's draft report in commit 73d76588e9664da54cd953e54f87b2edcfe2ec4e.

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

### 2.2.1. bcmsettings.sol - Revert transaction if losing the battle CRITICAL

In genBattle function, the user can generate the battle with a small amount less than the insurance fee. When he loses the battle, the transaction will be reverted, and he will lose nothing.

[redacted]

#### RECOMMENDATION

We should return the failed result instead of using require statement.

#### UPDATES

- *2021-10-03*: This issue has been acknowledged and fixed by the Blockchain Monster Hunt team.

### 2.2.2. bcmsettings.sol - User can skip generating the battle if the kill rate > 0 MEDIUM

Before generating the battle, the user can extract the kr value from the d1 parameter. If the kr value is greater than 0, they can skip creating this transaction and vise versa (to ensure they always win the battle).

[redacted]

#### RECOMMENDATION

Kill rate should be combined with a random value so that user cannot know the result before creating the transaction.

#### UPDATES

- *2021-10-03*: This issue has been acknowledged by the Blockchain Monster Hunt team. Based on their response, the battle logic is quite complex to be handled on-chain. Moreover, the backend already limits the number of battle requests from users, and users also need to battle to gain more exps. So, the fix for this issue has been skipped right now.

### 2.2.3. bcmsettings.sol - Weak random number generator HIGH

Current random number generator function which is used for the genGene and genBattle function is quite weak. For example, when this function is used in the genGene function, the output only depends on msg.sender, block.timestamp, and block.difficulty. However, the block timestamp in the BSC chain is quite easy to predict and the block difficulty is a constant also.

[redacted]

#### RECOMMENDATION

We should combine both blockhash and block timestamp when generating the random output. However, we should not rely on the output of this function much, rate-limiting the function calls using this random function from the backend is necessary.

[redacted]

#### UPDATES

- *2021-10-03*: This issue has been acknowledged and fixed by the Blockchain Monster Hunt team.

### 2.2.4. bcmerc721.sol - Missing non-contract call checking CRITICAL

In BlockchainMonster contract, the catchMonster and battleMonster functions are missing non-contract call checking. Consider an example with the catchMonster function, if an attacker wants to catch a monster without failure, he just needs to create a contract that makes a call to the BlockchainMonster contract. In this transaction, the value of block.timestamp can easily be read so that the result of the _random function in the BCMSettings can easily be guessed.

[redacted]

#### RECOMMENDATION

Adding onlyNonContract modifier to all functions which use the _random function.

[redacted]

#### UPDATES

- *2021-10-03*: This issue has been acknowledged and fixed by the Blockchain Monster Hunt team.

## 2.3. Additional notes and recommendations

### 2.3.1. bcmcerc20.sol - Unnecessary checks INFORMATIVE

The transfer function already check the balance before transferring, so the check require(balanceOf(msg.sender) >= fee, "low_balance") in the portMonsterToken function is unnecessary. There is an unnecessary check in the catchMonsterByBCMC function also.

[redacted]

**RECOMMENDATION**

Remove all the mentioned unnecessary checks.

**UPDATES**

- *2021-10-03*: This issue has been acknowledged and fixed by the Blockchain Monster Hunt team.

### 2.3.2. bcmtrade.sol - Unnecessary check in the executeTrade function INFORMATIVE

The totalFee is calculated from the price, but the price is checked with msg.sender already. So, the check require(totalFee < msg.value, "invalid_fee") here is unnecessary.

[redacted]

**RECOMMENDATION**

Remove the unnecessary check.

**UPDATES**

- *2021-10-03*: This issue has been acknowledged and fixed by the Blockchain Monster Hunt team.

## 2.4. Compatible chains

All the contracts providing in this audit is compatible with EVM-based chains, including Binance Smart Chain, Ethereum, Polygon, Huobi ECO Chain and Blockchain Monster Hunt's own chain Ambros.

# 3. VERSION HISTORY

| Version | Date | Status/Change | Created by |
|---|---|---|---|
| **1.0** | *2021-10-05* | Private Report | Verichains Lab |
| **1.1** | *2021-10-05* | Public Report | Verichains Lab |
| **1.2** | *2021-10-12* | Public Report | Verichains Lab |
| **1.3** | *2021-10-30* | Public Report | Verichains Lab |

*Table 2. Report versions history*