verichains

SECURITY AUDIT OF

AMAKUNI CORE GAME

Public Report

Oct 9, 2024

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.
ERC721	The ERC-721 introduces a standard for NFT, in other words, this type of Token is unique and can have different value than another Token from the same Smart Contract, maybe due to its age, rarity or even something else like its visual.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Oct 9, 2024. We would like to thank the Kuni Foundation 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 Amakuni Core Game. 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 Amakuni Core Game

AMAKUNI is a fully on-chain P2E action RPG with a fixed token supply and fair launch & fair distribution model.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the Kuni Protocols.

It was conducted on commit 94d1b64364388ae53ce04430a12d1c96bc7b0622 from git repository *https://github.com/kunifoundation/kuni-protocol*

The latest version of the following files were made available in the course of the review (commit 89b1e5ea0522b287c3e11242fb24e3eaed74ec67):

SHA256 Sum	File
2faa36f044bd1f566018783f891bc48081965be50ffe399f66e38fbd2d841bff	contracts/commons/EcoGame.sol
52e18895fade8a63ddb90de99c065f4814306cb4a1a36f8c4f01fbc0ca3ebcc1	contracts/commons/MetaData.sol
13c3f5300c88f2b918e9c643255139092f46c08e2ce628fa3ac84a3b650f20bf	contracts/game/AmaGame.sol
23bac8c47b5106b5066c362af92faec8a0065129ee01d111ba9617edfe6eabe4	contracts/game/AmaInv.sol
deca775168566ce48a4a3595487276bd2cf2ac2b38dfc876a69975dce7d2bc5b	contracts/game/MiningKuni.sol
94fd0ff41c411850bd96213b10f6316768705d23f20ca665a0a6dba1785957f9	contracts/game/Referral.sol
bb098e61843d260d1aaf3cde7973c4849da394b1a14da8463217c2528cb2ef1e	contracts/game/Scholarship.sol
505ad29b85dbee37c60e5fda087dc2d57ef6730b2bf80588cf09b13b11c25d4b	contracts/interfaces/IAmaGame.sol
fb6e65b66e23e366d46bfdada3f1dd5b8feeed53c9d70892f3f503fa532ebbec	contracts/interfaces/IAmaInv.sol

2080144e7fe9115f1b351739d28f34d04d0bab14c0b44ababd7c689c3ee0837d	contracts/interfaces/IData.sol
fa4807a3e59a0ae790852c188d578ea3431887eeadff4e2a08186f3f1557d24f	contracts/interfaces/IEcoGame.sol
49099a9a4fc7b0ca16beea01d9dbf267f936ea2afe5636fc964b30f9dc7792cc	contracts/interfaces/IERC20Burnable.sol
5adbe1ca20e0fdf97a39cc08b7d28635da285ac2e20f84cddd8c1d1ceef5aca3	contracts/interfaces/IERC20Mint.sol
a5feed4fa02b0b55f08a7bdbb71a51d882e344aef04efd3c52c3848c0916732e	contracts/interfaces/IERC721Mint.sol
1b47bdeae48eabd0af75189e3fdd25694dd08cc463cd03e36fbbb43252b771d3	contracts/interfaces/IMaterial.sol
08ca5ce71a29c88f860459680477946ca6f3ce7264cc53f8738b4b3f943a763b	contracts/interfaces/IMetaData.sol
dec772c05af609de178e58a0da9c71a52819a9fe64dfa5bc1e507ebabfcd62e2	contracts/interfaces/IMiningKuni.sol
baaa8034132c52ba431c8f685dee1562155fcd4483d433a3881213ce1780db1d	contracts/interfaces/IReferral.sol
8ca7231ab4fe850835d96d8be25a60e795ab7053d17605cf876fe8bf850e1b40	contracts/interfaces/IScholarship.sol
d1c7f076df3c89be63b337c439a9c697509b37955a61d3826e089cd13057ff6d	contracts/interfaces/IStoreGame.sol
86e652e782414382579ac3a90bcc9b8378b724879942a69995fbafa4403e2586	contracts/materials/GreenEnergy.sol
414bcac3db4c2afa58d60f38d0c6812b9bcf63d9e42eb7eba0e6e106d6913e1b	contracts/materials/Material.sol
e0fb7d1eaf09338ef35761a39e51557864f870fb84b15c337bbd584253935acc	contracts/nfts/KuniItem.sol
40cf660afb4a063eb5da9b04b4a2e1fef58f084ececea18bc306472bafa01392	contracts/nfts/KuniSaru.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

Kuni Foundation acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Kuni Foundation understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Kuni Foundation agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Kuni Foundation will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Kuni Foundation, the final report will be considered fully accepted by the Kuni Foundation without the signature.

2. AUDIT RESULT

2.1. Overview

The *kuni-protocol* are written in Solidity language and utilize `OpenZeppelin`'s contract libraries, emphasizing modularity and security.

2.1.1. Materials contracts

Green Energy and Material contract are ERC20-based contracts. Besides the default ERC20 functions, the Material and Green Energy contracts implement extend modifiers and some functions for `minter` role.

Green Energy is the reward which players will receive after each battle. Green Energy is used to increase Hashing Power of Miner building.

The table below lists some properties of the audited `GreenEnergy` contract (as of the report writing time).

PROPERTY	VALUE
Name	Green Energy
Symbol	GE
Total Supply	Unlimited

2.1.2. NFT contracts

Saru NFT contract is an ERC721 contract that extends `ERC721`, `ERC721Enumerable`, `Pausable`, `Ownable` and `Counters` contracts. Most of functions in `KuniSaru` contract are for `ADMIN_ROLE`. Besides, in NFT item contract, there is a `safeMint()` which can function can be using by `minter` role.

2.1.3. Game contracts

Core game contracts built around a play-to-earn game concept. The system includes reward-sharing mechanisms, staking management features, and interactions with various components such as referral systems, scholarships, and in-game items.

`MiningKuni` combines an ERC20 token with gas usage-based mining rewards. The system allows users to mine Kuni tokens based on the amount of gas they have consumed in transactions within certain supported pools, and it ensures secure interaction using mechanisms like reentrancy guard, ownership controls, and more.

`AmaInv` is an inventory management system for crafting items in a game-like environment. It integrates with an external mining contract, an economic game contract, and an ERC721 contract to allow users to craft items using various materials.

- Crafting Flow: Users can provide multiple materials for crafting, and the `EcoGame` contract handles the creation of the new item based on these materials and the user's cap.
- Cap Increment: Each user's crafting ability increases after every craft, allowing them to craft better or more items in subsequent operations.

The system also implements a referral system where users can create and apply referral codes to earn rewards. It features code creation, application, and rewards distribution based on the codes used.

`Scholarship` contract is a mechanism for NFT lending with rate management and batch operations, all of them are handled securely through reentrancy protection and ownership checks. Users can lend their NFTs with a certain rate and allow others to use these NFTs. Those scholarships later can be canceled or transferred to another user.

2.2. Findings

During the audit process, the audit team had identified some vulnerable issues in the contract code.

#	Issue	Severity	Status
1	Wrong logic in `_transferToken()`	MEDIUM	Acknowledged
2	Weak randomness in `advantagePoint()`	MEDIUM	Acknowledged
3	Missing requirement in `mint()` function	MEDIUM	Acknowledged
4	Unclear logic in `_primaryStat()`	INFORMATIVE	Acknowledged
5	Missing check for return value in `deposit()`	INFORMATIVE	Fixed
6	Redundant usage of *SafeMath* library	INFORMATIVE	Acknowledged
7	Wrong documentations for some functions	INFORMATIVE	Acknowledged

2.2.1. MEDIUM - Wrong logic in `_transferToken()`

Position:

- contracts/game/AmaGame.sol

The `_transferToken()` function currently reduces the token amount if the wallet balance is insufficient and does not check whether the transaction succeeded. However, this issue is unlikely to have a significant impact since most transactions involve sufficient token balances. While a user may receive fewer tokens than requested, it doesn't pose a critical risk to the user's assets or system operations.

```
function _transferToken(address mToken, uint256 amount) internal {
    uint256 canTransf = IERC20(mToken).balanceOf(address(this));
    if (amount > canTransf) {
        amount = canTransf;
    }
    IERC20(mToken).transfer(msg.sender, amount);
}

function withdraw() external override nonReentrant {
    // ...

    if (kuniStakedOf[msg.sender] > 0) {
        // @auditor: no reward for user if the wallet is exhausted
        _transferToken(miningKuni, kuniStakedOf[msg.sender]);
        // @auditor: but it doesn't revert and user's ballance is reset anyway
        kuniStakedOf[msg.sender] = 0;
    }
}
```

RECOMMENDATION

Add transaction checks and handle errors appropriately, or notify the user in cases of insufficient funds to ensure transaction integrity.

UPDATES

- ***Sep 25, 2024***: The issue is acknowledged by Kuni Foundation.

2.2.2. MEDIUM - Weak randomness in `advantagePoint()`

Position:

- contracts/commons/EcoGame.sol

The `advantagePoint()` function uses block properties (such as `block.difficulty` and `block.timestamp`) to determine the outcome of a match when two players have nearly equal stats. Although these properties can be manipulated by users under specific conditions, the potential

impact is minor as it only affects a narrow set of competitive situations and does not significantly disrupt the core mechanics of the game.

```
function advantagePoint() {
    // ...
    if (rate < 500) {
        won =
            uint256(keccak256(abi.encodePacked(block.difficulty, block.timestamp,
block.number, block.coinbase))).mod(100) >
            50;
    }
}
```

RECOMMENDATION

Improving randomness by integrating more reliable random sources is optional, as the issue has a minimal effect on the overall system.

UPDATES

- ***Sep 25, 2024***: The issue is acknowledged by Kuni Foundation.

2.2.3. MEDIUM - Missing requirement in `mint()` function

Positions:

- contracts/materials/Material.sol

The `mint()` function currently lacks a check for the token amount, which could potentially exceed the MAX_SUPPLY limit. However, since the token amounts are pre-calculated by `getRewardForMiner()`, the likelihood of surpassing the supply cap is low. Therefore, this issue poses minimal risk to the game's token economy.

```
function mint(address to, uint256 amount) external override onlyMinter {
    _mint(to, amount); // @auditor: amount is not restricted and may exceed MAX_SUPPLY
}
```

RECOMMENDATION

Adding a validation check is a minor improvement to ensure accuracy, but the issue does not have a significant impact on the system's functionality.

UPDATES

- ***Sep 25, 2024***: This issue has been acknowledged by Kuni Foundation.

2.2.4. INFORMATIVE - Unclear logic in `_primaryStat()`

Location:

- contracts/commons/EcoGame.sol

`_primaryStat()` is used by `_toCraftNameCat()` to determine primary stat of a item then loop up and return item's name together with item's category.`_primaryStat()` ignore the last stat of item, so the last stat can never be the primary stat. This can create a item with secondary stat has higher value than primary stat.

```
function _primaryStat(uint256[] memory items) internal pure returns (uint256 m1) {
    m1 = 0;
    for (uint256 inx = 1; inx < items.length - 1; inx++) { // @auditor: last stat is
ignored
        if (items[m1] < items[inx]) {
            m1 = inx;
        }
    }
}
```

RECOMMENDATION

Refactor data structure or provide a detail documentation for this implementation.

UPDATES

- ***Sep 25, 2024***: Kuni Foundation responded that the final stat cannot be the primary stat of an item, as this is part of the game's design. Therefore, the logic in the `_primaryStat()` function is correct.

2.2.5. INFORMATIVE - Missing check for return value in `deposit()`

Position:

- contracts/game/AmaGame.sol

When transferring Kuni from the user to the contract, the `deposit()` function will increase the user's balance regardless of whether the transaction is successful. Users can exploit this failure to manipulate their's balance and claim more rewards than they are entitled to.

```
function deposit(uint256 kuniAmount, uint256[] calldata tokenIds) external override
nonReentrant {
    // ...

    if (kuniAmount > 0) {
        // @auditor: this call may fail
        IERC20(miningKuni).transferFrom(msg.sender, address(this), kuniAmount);
        // @auditor: but user's ballance always increase
```

```
        kuniStakedOf[msg.sender] = kuniStakedOf[msg.sender].add(kuniAmount);
    }

    // ...
}
```

RECOMMENDATION

Check the return value of `transferFrom()` and handle errors as necessary. Alternatively, use the `SafeERC20` interface to automatically revert the transactions if they fail.

UPDATES

- ***Sep 25, 2024***: This issue has been acknowledged and fixed by Kuni Foundation.

2.2.6. INFORMATIVE - Redundant usage of SafeMath library

Positions:

- contracts/game/AmaGame.sol
- contracts/materials/Material.sol
- contracts/commons/MetaData.sol
- contracts/game/AmaInv.sol
- contracts/commons/EcoGame.sol
- contracts/game/MiningKuni.sol
- contracts/game/Referral.sol
- contracts/game/Scholarship.sol

From Solidity version 0.8, the compiler has built in overflow checking, so SafeMath is no longer needed.

RECOMMENDATION

Remove the SafeMath library and refactor the code to enhance its readability.

UPDATES

- ***Sep 25, 2024***: This issue has been acknowledged by Kuni Foundation.

2.2.7. INFORMATIVE - Wrong documentations for some functions

During audit process, we find out that some functions have wrong documentations:

- In file contracts/commons/EcoGame.sol, function `battleBonusInc()`: the k factor is inconsistent with within two lines.

- In file contracts/commons/MetaData.sol: function `addNft()` and `addNftBatch()` should swap comments to each other.

RECOMMENDATION

Double-check the documentation and correct them to prevent misunderstanding.

UPDATES

- ***Sep 25, 2024***: This issue has been acknowledged by Kuni Foundation.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Sep 25, 2024*	Public Report	Verichains Lab
1.1	*Sep 30, 2024*	Public Report	Verichains Lab
1.2	*Oct 9, 2024*	Public Report	Verichains Lab

Table 2. Report versions history