Smart Contract Audit Report

for

Tastemakerz

TRUSTLOOK

Version 0.1

Trustlook Blockchain Labs

Email: bd@trustlook.com

TRUSTLOOK

Project Overview

Project Name	Tastemakerz
Contract codebase	N/A
Platform	EVM compatible blockchains
Language	Solidity
Submission Time	2023.02.22

Report Overview

Report ID	TBL_20230222_00
Version	1.0
Reviewer	Trustlook Blockchain Labs
Starting Time	2023.02.22
Finished Time	2023.02.24

Disclaimer

Trustlook audit reports do not provide any warranties or guarantees on the vulnerability-free nature of the given smart contracts, nor do they provide any indication of legal compliance. The Trustlook audit process is aiming to reduce the high level risks possibly implemented in the smart contracts before the issuance of audit reports. Trustlook audit reports can be used to improve the code quality of smart contracts and are not able to detect any security issues of smart contracts that will occur in the future. Trustlook audit reports should not be considered as financial investment advice.

About Trustlook Blockchain Labs

Trustlook Blockchain Labs is a leading blockchain security team with a goal of security and vulnerability research on current blockchain ecosystems by offering industry-leading smart contracts auditing services. Please contact us for more information at (https:/www.trustlook.com/services/smart.html) or Email (bd@trustlook.com)

The Trustlook blockchain laboratory has established a complete system test environment and methods.

Black-box Testing	The tester has no knowledge of the system being attacked. The goal is to simulate an external hacking or cyber warfare attack.
White-box Testing	Based on the level of the source code, test the control flow, data flow, nodes, SDK etc. Try to find out the vulnerabilities and bugs.
Gray-box Testing	Use Trustlook customized script tools to do the security testing of code modules, search for the defects if any due to improper structure or improper usage of applications.

Introduction

By reviewing the smart contract's implementation, this audit report has been prepared to discover potential issues and vulnerabilities of their source code. We outline in the report about our approach to evaluate the potential security risks. Advice to further improve the quality of security or performance is also given in the report.

About Tastemakerz

Have fun while learning and earning your way to become a Web 3 Tastemakerz through a decentralized education guild. The project is backed by Animoca subsidiary, Forj with a mission to drive true Web 3 adoption.

About Methodology

To evaluate the potential vulnerabilities or issues, we go through a checklist of well-known smart contracts related security issues using automatic verification tools and manual review. To discover potential logic weaknesses or project specific implementations, we thoroughly discussed with the team to understand the business model and reduce the risk of unknown vulnerabilities. For any discovered issue, we might test it on our private network to reproduce the issue to prove our findings.

The checklist of items is shown in the following table:

Category	Type ID	Name	Description
Coding Specification	CS-01	ERC Standards	The contract is using ERC standards.
	CS-02	Compiler Version	The compiler version should be specified.
	CS-03	Constructor Mismatch	The constructor syntax is changed with Solidity versions. Need extra attention to make the constructor function right.
	CS-04	Return standard	Following the ERC20 specification, the transfer and approve functions should return a bool value, and a return value code needs to be added.
	CS-05	Address(0) Validation	It is recommended to add the verification of require(_to!=address(0)) to effectively avoid unnecessary loss caused by user misuse or unknown errors.
	CS-06	Unused Variable	Unused variables should be removed.
	CS-07	Untrusted Libraries	The contract should avoid using untrusted libraries, or the libraries need to be thoroughly audited too.
	CS-08	Event Standard	Define and use Event appropriately
	CS-09	Safe Transfer	Using safeTransfer/transfer to send funds instead of send.
	CS-10	Gas Consumption	Optimize the code for better gas consumption.
	CS-11	Deprecated Uses	Avoid using deprecated functions.
	CS-12	Sanity Checks	Sanity checks when setting key parameters in the system
	CS-13	Typo	Typo in comments or code
	CS-14	Fallback Function	Splitting fallback and receive function
	CS-15	Comment Standard	Use clear consistent comments with code semantics
	CS-16	Naming Standard	Use standard method to name functions and variables

Coding Security	SE-01	Integer overflows	Integer overflow or underflow issues.
	SE-02	Reentrancy	Avoid using calls to trade in smart contracts to avoid reentrancy vulnerability.
	SE-03	Transaction Ordering Dependence	Avoid transaction ordering dependence vulnerability.
	SE-04	Tx.origin usage	Avoid using tx.origin for authentication.
	SE-05	Fake recharge	The judgment of the balance and the transfer amount needs to use the “require function”.
	SE-06	Replay	If the contract involves the demands for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks.
	SE-07	External call checks	For external contracts, pull instead of push is preferred.
	SE-08	Weak random	The method of generating random numbers on smart contracts requires more considerations.
Additional Security	AS-01	Access control	Well defined access control for functions.
	AS-02	Authentication management	The authentication management is well defined.
	AS-03	Semantic Consistency	Semantics are consistent.
	AS-04	Functionality checks	The functionality is well implemented.
	AS-05	Business logic review	The business model logic is implemented correctly.

The severity level of the issues are described in the following table:

Severity	**Description**
Critical	The issue will result in asset loss or data manipulations.
High	The issue will seriously affect the correctness of the business model.
Medium	The issue is still important to fix but not practical to exploit.
Low	The issue is mostly related to outedate, unused code snippets.
Informational	This issue is mostly related to code style, informational statements and is not mandatory to be fixed.

Audit Results

The Trustlook security team has used the team's analysis tools and manual audit process to audit the project. No obvious risks were identified during the audit. There are some comments and some enhancement suggestions in the following sections.

Scope

Following files have been scanned by our internal audit tool and manually reviewed and tested by our team:

File names	Sha1
ERC1155SelfMinter.sol	5e236b066848b188312e1d4887e515d64ff9323e

Summary

Issue ID	Severity	Location	Type ID	Status
TBL_SCA_001	HIGH	ERC1155SelfMinter.sol:332	AS-01	Fixed
TBL_SCA_002	MEDIUM	ERC1155SelfMinter.sol:250 ERC1155SelfMinter.sol:256	AS-04	Fixed
TBL_SCA_003	LOW	ERC1155SelfMinter.sol:394	AS-05	Fixed
TBL_SCA_004	LOW	ERC1155SelfMinter.sol:85 ERC1155SelfMinter.sol:93	AS-04	Fixed
TBL_SCA_005	INFO	ERC1155SelfMinter.sol:49 ERC1155SelfMinter.sol:52	CS-10	Fixed
TBL_SCA_006	INFO	ERC1155SelfMinter.sol:48 ERC1155SelfMinter.sol:51	CS-15	Fixed

Details

• ID: TBL_SCA-001

• Severity: HIGH

• Type: AS-01 (Access control)

• Description:

The function *burn()* is set as public and anyone can burn NFT tokens for any address *from*. Players's tokens could be burned by anonymous users.

• Remediation:

This issue has been fixed in a new release.

• ID: TBL_SCA-002

• Severity: MEDIUM

• Type: AS-04 (Functionality checks)

• Description:

The functions *setTierMerkleRoots()* and *setLimitPerWalletPerTier()* set the new tiers data starting from index 0. However, the array is starting from 1.

• Remediation:

This issue has been fixed in a new release.

• ID: TBL_SCA-003

• Severity: LOW

• Type: AS-05 (Business logic review)

• Description:

The function *_checks()* is used to validate the payment fee for minting new NFT tokens. However, it seems the payment fee is not dependent on the *amount*.

It is recommended to design a better fee calculation logic for minting specific amounts of NFT tokens.

• Remediation:

This issue has been fixed in a new release.

• ID: TBL_SCA-004

• Severity: LOW

• Type: AS-04 (Functionality checks)

• Description:

The members *num* and *walletAddress* are not necessary to be included in the structures *Tier* and *Partner* since they are used as the key for the mappings *tiers* and *partners*.

• Remediation:

The issue has been partially fixed in a new revision.

• ID: TBL_SCA-005

• Severity: INFO

• Type: CS-10 (Gas consumption)

• Description:

The mappings *maxSupplyPerId* and *totalSupplyPerId* are defined to use the same key and both used at the same locations.

It is recommended to merge them to mapping (uint256 => Supply) and define a data structure *Supply* to have *max* and *total*. In this way the mapping calculation for both variables could be simplified and gas consumption is reduced.

• Remediation:

This has been fixed in a new release.

• ID: TBL_SCA-006

• Severity: INFO

• Type: CS-15 (Comment standard)

• Description:

The comments for the mappings *maxSupplyPerId* and *totalSupplyPerId* are not consistent with the code. It should be stated as follows:

// TokenId => Max Supply
// TokenId => Total Supply

• Remediation:

This has been fixed in a new release.