Smart Contract Audit Report

for

Kepler

TRUSTLOOK

Version 0.1

Trustlook Blockchain Labs

Email: bd@trustlook.com

TRUSTLOOK

Project Overview

Project Name	Kepler
Contract codebase	N/A
Platform	Avalanche/BSC/Ethereum/Polygon
Language	Solidity
Submission Time	2022.04.06

Report Overview

Report ID	TBL_20220406_00
Version	1.0
Reviewer	Trustlook Blockchain Labs
Starting Time	2022.04.06
Finished Time	2022.04.18

@ Copyright 2022 Trustlook - All rights reserved

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.

@ Copyright 2022 Trustlook - All rights reserved

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.

@ Copyright 2022 Trustlook - All rights reserved

Introduction

By reviewing the implementation of Kepler's smart contracts, 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 Kepler

Kepler's long-term vision includes: bringing more players into the NFTs game world, returning to the fun of the game itself, and influencing more game content creators. Facing the unknown future, Kepler is not limited to the RPG game itself, we believe the world of Kepler has thousands of possibilities.

Kepler is a 3D Sci-Fi MMORPG based on multiple public chains, combining real-time PVE/PVP combat, player socialization, simulation, and so on.

Each asset in the Kepler game world is a separate NFT, and players can earn and hold their assets in the game world. These assets can be freely traded through the Kepler Market, Opensea, or other marketplaces.

In the game, players take on the role of a member of the interstellar migrant fleet, carving out a world of their own by exploring the planet Kepler and interacting with other players.

@ Copyright 2022 Trustlook - All rights reserved

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	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

@ Copyright 2022 Trustlook - All rights reserved

	CS-16	Log Function	Log functions should be removed in production code
	CS-17	Duplication	Duplicated function, variable, structure.
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.
	AS-06	Unrestricted Function	User should be aware of unrestricted function

@ Copyright 2022 Trustlook - All rights reserved

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.

@ Copyright 2022 Trustlook - All rights reserved

Audit Results

Here are the audit results of the smart contracts.

Scope

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

File names	Sha1
nft/NFTMarket.sol	49785d77ae4fc0baefd2df40a4b042e87b0925a3
nft/KeplerNFT.sol	da1f7f36bc6be8209bf8b005adbeab5ffaa984b5
nft/MysteryBox.sol	4ae4c692fe5fd7769d419a9f69f388208d379a2c
nft/NFT.sol	487e37c333e239871fc03b635e72f899d1c704f8
nft/interfaces/IKeplerNFT.sol	3f8f02ba7264aa76ee8dae3edf02aaa426630adb
nft/interfaces/INFTMarket.sol	c369e3203da24ecaa9ec4d846cf06fe10d2aade3
nft/interfaces/IMysteryBox.sol	396e01c16a520061f1a24fef96e0d837df5a63ff
bridge/Bridge.sol	cc1d57958f224a41b8901a8fec836b32673376f6
bridge/interfaces/IBridge.sol	f5b8556753383534895d4b5af82da85fca0c8e2d
oracle/Oracle.sol	80d2b3a5c8c77addd7000edb0e060ef245c96725
oracle/interfaces/IOracle.sol	78b32ce9a1e77e608f20c535526e8c4aceae9ce3
libraries/SafeDecimalMath.sol	261c1e3c0316a919bdc3bcdb56199a8d71627959
libraries/Signature.sol	5c7a06392781a3283e7ccc17a17365f0709f14f5
common/Minable.sol	0b75ae7913fd0e839c25405541d636ee00a5d4e6
pool/RewardPool.sol	8df261db8ce4f9bbd45a4cbe121bb2b88d73150e
pool/DepositPool.sol	b8e813696ab67bd75eb89e9c68bf0662dc3e1ead
pool/CorePool.sol	fcc336f317bccb6be15adf3caab6ec611903035b
pool/PoolFactory.sol	4f4284b8b781fbd66474dac93d76d53a1d38c527
pool/BasePool.sol	1de0a242380b32962d80c59b3a910cda11a82b34
pool/interfaces/ICorePool.sol	5ac0d444b7e5883178c95f26c7fdfbb770ab4fe8
pool/interfaces/IRewardPool.sol	1297d51b301013ebcad14587dd9e2c163c1a0bc5
pool/interfaces/IPoolFactory.sol	9f8efe899fee9e3308759ee263e0d276cad6d595
tokens/Token.sol	11618d310a2f5f3d117908c912d4daa579e73f80
tokens/interfaces/IToken.sol	765cb4b01d6f8af5baf176776229c0f0cff6e675

@ Copyright 2022 Trustlook - All rights reserved

Summary

Issue ID	Severity	Location	Type ID	Status
TBL_SCA_001	Info	Oracle.sol:26	AS-06	Closed
TBL_SCA_002	Low	Oracle.sol:29	CS-12	Fixed
TBL_SCA_003	Info	BasePool.sol	CS-06	Fixed
TBL_SCA_004	Info	BasePool.sol:10	CS-17	Fixed
TBL_SCA_005	Low	PoolFactory.sol:73	CS-12	Fixed
TBL_SCA_006	Info	CorePool.sol:4	CS-16	Fixed
TBL_SCA_007	Info	CorePool.sol:29	CS-13	Fixed
TBL_SCA_008	Medium	CorePool.sol:129	AS-05	Closed
TBL_SCA_009	Medium	CorePool.sol:239	AS-05	Fixed
TBL_SCA_010	Info	Token.sol:22	AS-06	Closed
TBL_SCA_011	Info	MysteryBox.sol:58,65,69	CS-08, CS-12	Fixed
TBL_SCA_012	Info	MysteryBox.sol:177	CS-10	Fixed
TBL_SCA_013	Medium	MysteryBox.sol:324	AS-05	Fixed
TBL_SCA_014	Info	NFTMarket.sol:3	CS-06	Fixed
TBL_SCA_015	Info	NFTMarket.sol:86	CS-05	Fixed
TBL_SCA_016	Medium	NFTMarket.sol:235	AS-05	Fixed
TBL_SCA_017	Info	Bridge.sol: 35, 395, 419	CS-05	Fixed
TBL_SCA_018	Info	RewardPool.sol: 48, 49	CS-12	Fixed
TBL_SCA_019	Medium	RewardPool.sol:63	AS-05	Fixed

@ Copyright 2022 Trustlook - All rights reserved

Details

- ID: TBL_SCA_001
- Severity: Info
- Location: Oracle.sol:26

Type: AS-06 (Unrestricted Function)

- Description:

  The asset prices can be updated with no restrictions. Users should be aware of this risk.

- Remediation:

  The Kepler team is aware of this and will update the Oracle.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_002

• Severity: Low

• Location: Oracle.sol:29

Type: CS-12 (Sanity Checks)

- Description:

  It is recommended to check length for both *assets* and *prices.*

- Remediation:

  This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_003

• Severity: Info

• Location: BasePool.sol

Type: CS-06 (Unused)

• Description:

The code is not used. It is recommended to remove the unused code.

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_004

• Severity: Info

• Location: BasePool.sol:10

Type: CS-17 (Duplication)

• Description:

StakingItem and *UserStakingItem* are duplicated.

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_005

• Severity: Low

• Location: PoolFactory.sol:73

Type: CS-12 (Sanity Checks)

• Description:

It is recommended to check length for both *pools* and *weights.*

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_006

• Severity: Info

• Location: CorePool.sol:4

Type: CS-16 (Log Function)

- Description:

  It is recommended to remove console.log functions in production code.

- Remediation:

  This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_007

• Severity: Info

• Location: CorePool.sol:29

Type: CS-13 (Typo)

• Description:

Variable *depoistAmount* is a typo, should be *depositAmount*. Please note that another variable also named *depositAmount* is used at line 233.

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_008

• Severity: Medium

• Location: CorePool.sol:129

Type: AS-05 (Business Logic)

• Description:

If function *updateLockUnitMultiplier* is called and *extraWeightedAmount* is updated using the new *lockUnitMultiplier*, then each deposit in *_userDeposits* should also update the *deposit.extraWeightedAmount*.

• Remediation:

The Kepler team explained that once a user has staked, the user*Deposit.extraWeightedAmount* will never be changed. Function *updateLockUnitMultiplier* only affects future deposits.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_009

• Severity: Medium

• Location: CorePool.sol:239

Type: AS-05 (Business Logic)

• Description:

extraWeightedAmount -= extraWeightedAmount;
should be updated to
extraWeightedAmount -= deposit.extraWeightedAmount;

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_010

• Severity: Info

• Location: Token.sol:22

Type: AS-06 (Unrestricted Function)

• Description:

Owner can mint tokens to any address. Users should be aware of this risk.

• Remediation:

The Kepler team will introduce multisig from Gnosis to help alleviating the risk.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_011

• Severity: Info

• Location: MysteryBox.sol: 58, 65, 69

Type: CS-08 (Event Standard), CS-12 (Sanity Checks)

• Description:

It is recommended to do sanity checks and emit events when updating system configurations.

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_012

• Severity: Info

• Location: MysteryBox.sol: 177

Type: CS-10 (Gas COnsumption)

• Description:

Ths if statement can be simplified as below:

```
if (price >= paymentConfig.maxPrice - paymentConfig.priceStep) {
        result = paymentConfig.maxPrice;
}
```

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_013

• Severity: Medium

• Location: MysteryBox.sol:324

Type: AS-05 (Business Logic)

• Description:

uint256 femaleCount = _genderTokenIds[MALE].length();
should be updated to
uint256 femaleCount = _genderTokenIds[FEMALE].length();

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA_014

• Severity: Info

• Location: NFTMarket.sol:3

Type: CS-06 (Unused)

• Description:

It is recommended to remove unused imports.

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA-015

• Severity: Info

• Location: NFTMarket.sol: 86

Type: CS-05 (Address Validation)

• Description:

It is recommended to validate that the address is not 0.

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA-016

• Severity: Medium

• Location: NFTMarket.sol: 235

Type: AS-05 (Business Logic)

• Description:

STATUS_OPEN should be *status*.

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA-017

• Severity: Info

• Location: Bridge.sol: 35, 395, 419

Type: CS-05 (Address Validation)

• Description:

It is recommended to validate that the address is not 0.

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA-018

• Severity: Info

• Location: RewardPool.sol: 48, 49

Type: CS-12 (Sanity Checks)

• Description:

It is recommended to validate that the parameter is not 0.

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved

• ID: TBL_SCA-019

• Severity: Medium

• Location: RewardPool.sol: 63

Type: AS-05 (Business Logic)

• Description:

The *_withdraw*() function lets a user withdraw his locked rewards. To get the full amount, a user has to withdraw 12 times with *withdrawInterval* between the withdrawals.

However, there are two issues in the current implementation. First, a user cannot wait 12 * *withdrawInterval* and do one withdrawal with the full amount.

Second, after the first *_withdraw()*, the *lockedReward.withdrawCount* is 1. To do a second *_withdraw()*, *maxWithdrawCount* needs to be at least 2, which means 2 * *withdrawInterval* of wait time. After the second *_withdraw(), lockedReward.withdrawCount* is 2. To do a third *_withdraw()*, *maxWithdrawCount* needs to be at least 3, which means 3 * *withdrawInterval* of wait time. This goes on until the last *_withdraw()*. To fix this problem, developer can remove

lockedReward.lastWithdrawTime=block.timestamp;

Or change

maxWithdrawCount > lockedReward.withdrawCount,

To

maxWithdrawCount >= 1

• Remediation:

This issue has been fixed in a new release.

@ Copyright 2022 Trustlook - All rights reserved