HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Kitsumon
Date: April 6th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Kitsumon.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC721 token;Transfer controller
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/Kitsumon/contracts/commit/31e7e70de90bf22fe033b02347c573484f202e69 - Initial Audit https://github.com/Kitsumon/contracts/tree/v2 - Connect to preview - remediation check https://github.com/Kitsumon/contracts/tree/v2 - remediation check 2
Commit	31e7e70de90bf22fe033b02347c573484f202e69 - Initial Audit 7f172b99aba09282031c94437d82b970b3b84340 - Remediation Check 2
Technical Documentation	Yes
JS tests	Yes
Website	https://kitsumon.com
Timeline	16 FEBRUARY 2022 - 06 APRIL 2022
Changelog	26 FEBRUARY 2022 - Initial Audit 14 MARCH 2022 - remediation check 06 APRIL 2022 - remediation check 2

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Kitsumon (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contract and its code review conducted between February 16th, 2022 - February 26th, 2022.

The second review was conducted on March 14th, 2022.

The remediation review was conducted on April 6th, 2022.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://github.com/Kitsumon/contracts/commit/31e7e70de90bf22fe033b02347c573484f202e69 - Initial Check
https://github.com/Kitsumon/contracts/tree/v2 - Remediation Check 2
Commit:
31e7e70de90bf22fe033b02347c573484f202e69 - Initial Audit
7f172b99aba09282031c94437d82b970b3b84340 - Remediation Check 2
Technical Documentation: yes
JS tests: Yes
Contracts:

TokenStaking.sol
PaymentAcceptor.sol
PaymentAcceptorFcfs.sol
KMCTGEVestingV1.sol
WalletUpgradeable.sol
TokenVestingUpgradeable.sol
AuctionHouse.sol
Market.sol
WhitelistUpgradeable.sol
WhitelistUpgradeableBase.sol
WhitelistCountertUpgradeable.sol
AuctionHouse.sol
Market.sol
MinterPresetUpgradeable.sol
SecurityPresetUpgradeable.sol
KMCV1.sol
ERC20CappedPresetUpgradeable.sol
Erc20PresetUpgradeable.sol
ERC721CappedPresetUpgradeable.sol
Erc721PresetUpgradeable.sol
EggRedemptionCardV1.sol
Potions.sol
RandomEggMinter.sol
ERC721KitsuMarkeplace.sol
ERC721CappedUpgradeable.sol
WhitelistUpgradeable.sol
WhitelistUpgradeableBase.sol
WhitelistCountertUpgradeable.sol

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are well-secured.

Our team analyzed code functionality, manual audit, and automated checks with Mythx and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **4** critical, **1** high, **4** medium, **12** low severity issues.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

As a result of the remediations review, Customers' smart contracts contain **2** critical, **1** high, **3** medium, and **12** low severity issues.

As a result of the remediations review, Customers' smart contracts contain **no** issues.

Notice: Tokenstaking.sol hints nothing whether it holds reward and stake balance together or separately. Check this logic and make sure they are separated

Notice: RandomEggMinter.sol uses chainlink VRF to generate random numbers. Its requestRandomness method may take fees. See the following issue: https://github.com/smartcontractkit/chainlink/pull/5943

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to asset loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to asset loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Audit overview

■■■■ Critical

1. Wrong Control Mechanism

   In the KMCv1.sol contract's setTradingStart() function, the required statement that checks the start time of trading can never be greater than the current time.

   Contracts:KMCv1.sol

   Function: setTradingStart() lines #52-57

   Recommendation: Convert this statement to check _time or set an initial value to _tradingStartTime and make condition >= 0

   Status: Fixed

2. Gas Limit not Set

   The Gas limit is not set when transferring Ether. If the underlying contract has a receive or fallback function that infinitely consumes gas (e.g. infinite loop) a bid could not be beaten, and execution of any function that calls the `_handleOutgoingBid` function will fail and consume enormous amounts of gas.

   Contracts: AuctionHouse.sol

   Function: _handleOutgoingBid()

   Recommendation: send only wrapped Ethereum or transfer eth via send function and validate its results.

   Status: Fixed

■■■ High

Potential Lock of Funds

In the _handeOutgoingBid() function, the failure condition of the Ether transfer is checked with an if statement. This can lead to the locking of funds. If the receiving contract is not able to handle incoming Ether

Contracts:AuctionHouse.sol

Function: _handleOutgoingBid() line #393

Recommendation: Change this control mechanism with a require statement.

Status: Fixed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Medium

1. Mismatching Calculations

   In the finalizeNFTTransfer() function, the formula for the bid share of the new owner is supposed to be 100 - creatordShare - sellOnShare, but in the below line, it only deducts creator share from 100.

   Contracts: Market.sol

   Function: _finalizeNFTTransfer()

   Recommendation: Review this logic.

   Status: Mitigated

2. Redundant Function Call

   In WhitelistCounterUpgradeable.sol contract, the function isWhitelistedAmt executes only if it is not paused, but there is a condition for if it is paused.

   Contracts:WhitelistCounterUpgradeable.sol

   Functions: isWhitelistedAmt()

   Recommendation: Remove the redundant.

   Status: Fixed

3. Unused Return

   In the ERC721KitsuMarketplace.sol contract, the removeBid function ignores the IMarket.removeBid()'s return value.

   Contracts:ERC721KitsuMarketplace.sol

   Function: removeBid()

   Recommendation: Implement Control Mechanisms.

   Status: Fixed

Low

1. Missing Checks.

In KMCv1.sol contract, unthrottleAccount function needs to check if the account is already unthrottled. Again in KMCv1.sol, the whitelistAccount() function needs to check if the account is already whitelisted.

Contracts:KMCv1.sol

Function: whitelistAccount(), unthrottleAccount()

Recommendation: Implement necessary control mechanisms.

Status: Fixed

2. Missing Whitelist Control

In WhitelistCounterUpgradeable.sol contract, the function isWhitelistedAmt() needs to check if the address is whitelisted first before whitelisting the amount.

Contracts:WhitelistCounterUpgradeable.sol

Functions: isWhitelistedAmt()

Recommendation: Implement necessary control mechanisms.

Status: Fixed

3. Function Calling Assumption.

In TokenStakingV1.sol contract, function __TokenStakingV1_init_unchained() function is created with the sole purpose of being called in __TokenStakingV1_init_(). However, its visibility is public so that it can be called by everybody anytime.

In PaymentAcceptor.sol contract, function __PaymentAcceptor_init_unchained() function is created with the sole purpose of being called in __PaymentAcceptor_init_(). However, its visibility is public so that it can be called by everybody anytime.

Contracts:TokenStakingV1.sol, PaymentAcceptor.sol

Function:

__PaymentAcceptor_init_unchained(), __TokenStakingV1_init_unchained()

Recommendation: Change function visibility.

Status: Fixed

4. Boolean Equality

Boolean constants can be used directly and do not need to be compared to true or false.

Contracts:AuctionHouse.sol, SecurityPresetUpgradeable.sol

Function: endAuction(), whenEnabled()

Recommendation: Remove boolean equality.

Status: Fixed

5. Missing Zero Address Validation.

In the Potions.sol contract, *the state* variable blacklist needs to be checked if it is 0x0.

Contracts: Potions.sol

Function: -

Recommendation: Implement zero address checks.

Status: Fixed

6. Redundant Calculation

In the endAuction() function, instead of calculating the end time over and over again. Call the getEndTime to save Gas

Contracts: AuctionHouse.sol

Functions: endAuction(uint256 auctionId)

Recommendation: Remove this statement.

Status: Fixed

7. Redundant Function

getCurrentTime() function only returns block.timestamp.

Contracts: TokenVestingUpgradeable.sol

Functions: getCurrentTime()

Recommendation: Remove this function.

Status: Fixed

8. Return Value Ignored

In PaymentAcceptor and *PaymentAcceptorFcfcs* contracts, the *pay* function uses *add()* to add elements into a set. However, the return value of this call is not checked.

Contracts: *PaymentAcceptor*.sol, *PaymentAcceptorFcfcs.sol*

Functions: *pay*()

Recommendation: Implement return value checks.

Status: Fixed

9. Smaller Variable Type is being Used

In TokenStakingV1.sol, the state variable *apy* is initialized as a uint256, but in the initializer functions, uint8 is passed to it.

Contracts:TokenStakingV1.sol

Functions: -

Recommendation: Adjust variable type accordingly.

Status: Fixed

10. Use of Hardcoded Values

__AuctionHouse_init_unchained() function uses hardcoded values 15 and 60 in multiplication

During the computation for the _maxTrnsferAMount variable, in KMCv1.sol, hardcoded values are being used .

Contracts: AuctionHouse.sol, KMCv1.sol

Functions: *__AuctionHouse_init_unchained*()

Recommendation: Move hardcoded values to constants.

Status: Fixed

11. Floating Pragma

Market.sol and AuctionHouse.sol contracts use floating pragma ^0.8.6.

Contracts: Market.sol and AuctionHouse.sol

Functions: -

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed

12. Functions that can be Declared as *external*

In order to save Gas, public functions that are never called in the contract should be declared as *external*.

Contracts:TokenStakingV1.sol,TokenVestingUpgradeable.sol, WalletUpgradeable.sol, *PaymentAcceptor*.sol, *PaymentAcceptorFcfcs.sol, KMCTGEVestingV1.sol*,Market.sol,AuctionHouse.sol,SecurityPresetUpgradeable.sol,MinterPresetUpgradeable.sol,KMCV1.sol,ERC20CappedPresetUpgradeable.sol,Erc20PresetUpgradeable.sol,ERC721CappedPresetUpgradeable.sol,Erc721PresetUpgradeable.sol,EggRedemptionCardV1.sol,ERC721KitsuMarkeplace.sol,ERC721CappedUpgradeable.sol

Functions:__TokenVestingUpgradeable_init(),__WalletUpgradeable_init(),__*PaymentAcceptor*_init(),__*PaymentAcceptorFcfs*_init(),__*KMCTGEVestingV1*_init(), __TokenStakingV1_init(),stake(),redeem(),setMaxAmount(),setMinAmount(),fund(),getStakeCount(),getStake(),getTotal(),getTotalRewardFund(),emergencyTokenDrain(),emergencyNftDrain(),emergencyDrain(),_AuctionHouse_init(address,address,address,address,uint256,uint8,address,__Market_init(address,address),__SecurityPresetUpgradeable_init(),__MinterPresetUpgradeable_init(),__KMCV1_init(),__ERC20CappedPresetUpgradeable_init(),__Erc20PresetUpgradeable_init(),__ERC721CappedPresetUpgradeable_init(),__Erc721PresetUpgradeable_init(),__EggRedemptionCardV1_init(),__ERC721KitsuMarkeplace_init(), __ERC721CappedUpgradeable_init()

Recommendation: Change function visibility.

Status: Fixed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.