HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Junkyard
Date: March 1, 2023

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 Junkyard
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	ERC721 system
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://junkyard.wtf/
Changelog	24.11.2022 - Initial Review 01.02.2023 - Second Review 16.02.2023 - Third Review 01.03.2023 - Fourth Review

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 Junkyard (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 contracts.

Scope

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

Initial review scope

Repository	https://github.com/BlackMagicCorp/jkd-contracts
Commit	13d31b439fbc698508954fd1b0b47515736e94dd
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/JKD.sol SHA3: 90703272b0703a35a63d285d0409e718865c93772c35547cda8dba10f731771d File: ./contracts/JKDManager.sol SHA3: d9ec312ab18d7a12d073d474deb780fe1371eb7f13178ef503034a4106e28cb1

Second review scope

Repository	https://github.com/BlackMagicCorp/jkd-contracts
Commit	a9576b1371919c2f24ead0266f002151d11c269e
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/Junkyard.sol SHA3: 440a56c0bf34a1488330024004b20a2c44a37dee321c10898da162ee5de1213f File: ./contracts/JunkyardManager.sol SHA3: 8bc617de3533a35c63fbe0067f28c6705f21b1bf1cae1201d770131ff4208fd5 File: ./contracts/JunkyardStorage.sol SHA3: e2ae02dace07664a8a597eefaafaf0588b518aed3fdfa8056d35e133bd656386

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

Third review scope

Repository	https://github.com/BlackMagicCorp/jkd-contracts
Commit	8a3fba537baa793ce2cc6f6768284d85e6b0c9e7
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/Junkyard.sol SHA3: bda079298759757058e442a64fee28fd3d482db25184985ef10e7ec064ad519f File: ./contracts/JunkyardManager.sol SHA3: e61252133cda1372d0fa140f91cbb78167308387bc6321fb05eff3e82126697c File: ./contracts/JunkyardStorage.sol SHA3: 865a9a1062a7846d85bd924d7dbca5cb9ac1a2ed9252a4bc17b9093575d1290b

Fourth review scope

Repository	https://github.com/BlackMagicCorp/jkd-contracts
Commit	7f018433e5e8370913abcabda2fcb89559388231
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/Junkyard.sol SHA3: bda079298759757058e442a64fee28fd3d482db25184985ef10e7ec064ad519f File: ./contracts/JunkyardManager.sol SHA3: e796f1709fe1eea82b2f3785d473aa4b3788ca0e1638b0ceb28f386ff0afc375 File: ./contracts/JunkyardStorage.sol SHA3: 805302aabe603cc399528e59258378273040c2154a2bd4ab9252594fe9bc5692

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to assets loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect the code quality

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

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

- A lite paper is provided with global functionalities
- Technical description is provided in ReadMe.
- NatSpec should be generated into contract documentation published on the website.
- The lite paper documents functionalities that are not implemented (JunkCoin launch, JunkCoin rewards, the Great Burn).

Code quality

The total Code Quality score is **10** out of **10**.

- Solidity official style guidelines are followed perfectly.
- The development environment is configured.

Test coverage

Test coverage of the project is **100%** (branch coverage).

- The code is well covered with tests.
- Only one negative case coverage is missing.
- Interactions by several users are not tested thoroughly.

Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.8**.

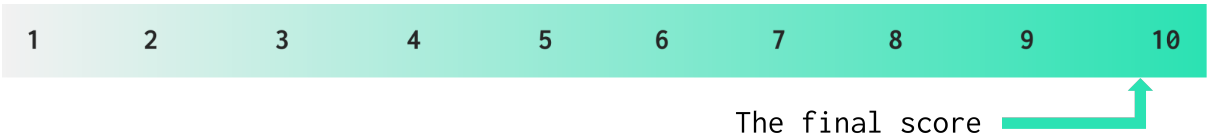

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
24 November 2022	7	2	5	3
1 February 2023	5	1	2	0

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

16 February 2023	3	0	1	0
1 March 2023	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Passed
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Not Relevant
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Not Relevant

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

Junkyard is a "dump to earn" platform where users can dump their useless NFT in exchange for native tokens (Junkcoin).
Users can pay (in ETH) to "fish" one or multiple of these NFT and choose one of them to claim.
It includes the following contracts:

- *Junkyard.sol - A contract where the users can interact to fish and claim NFT. It will then be bridged to Polygon contracts. This contract is on Ethereum.*
- *JunkYardManager.sol - A contract where all the logic and processes are done. This contract is on Polygon to reduce costs.*
- *JunkyardStorage.sol - A contract (on Ethereum) where the NFT are stored. Users can dump them here and receive them from here.*

Privileged roles

- The owner of Junkyard.sol can set new prices, set the manager address and chain, the storage address and chain, pause/unpause the contract.
- The owner of JunkyardStorage.sol can set the manager address and chain.
- The owner of JunkyardManager.sol can set Junkyard address and chain, set storage address and chain, set poop address and chain.
- Tha Admin role of JunkyardManager.sol can register a new token, register a new fishing attempt and register a collection.

Risks

- Some functionalities depend on **out-of-scope off-chain** management. These contracts do not provide any guarantees to users who interact with them.
- The **Junkcoin rewards for NFT dumping functionality described in the litepaper is not implemented** in the code yet. The coin itself is not implemented yet. There is no insurance that they will be in the future.
- The **"Great Burn" functionality described in the litepaper is not implemented** in the code yet. There is no insurance that it will be in the future.
- The **DAO described in the litepaper is not implemented** in the code yet. There is no insurance that it will be in the future.
- The dump process and the fishing process rely on the junkbot, which is an off-chain management system. **This off-chain system needs to be audited separately** as it is not in the scope of this audit.
- Without the junkbot, this smart contracts system can not work independently.
- The project uses Axelar as a gateway between Ethereum contracts and Polygon contracts. However, **Axelar contracts are not in the scope** of this audit.

- The function registerCollection loops on an array provided as a parameter by the owner. If the size of this array is not handled correctly, this can lead to a Gas limit excess. The owner should take it into account when calling this function.

Findings

■■■■ Critical

1. Highly Permissive Role Access

When the users use the *fishing* function, they have to pay, but this does not give them the possibility to claim their NFT. The owner of the contract must send them manually through the *claimLoot* function.

Path: ./contracts/JKD.sol

Recommendation: Owners should not have access to funds that belong to users.

Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

2. Requirements Violation

The documentation claims that everything happens on-chain.

“everything happens fully on-chain!”

“it's 100% on-chain fishing.”

This is not the case. Everything is done through off-chain management. The only thing that the user can do on-chain is to pay (through the fishing function) and access some getters. Everything else is in the hands of the owner.

Path: ./contracts/JKD.sol

Recommendation: The code should not violate the requirements provided by the Customer.

Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

3. Requirements Violation

“You will earn Junkcoin for each NFT you ditch to the Junkyard, the rarer the NFT is, the more Junkcoin you will earn.”

According to the documentation, users can send to the contract their own NFTs and receive as reward Junkcoin. Depending on how rare NFT is, more Junkcoin will be earned. Contracts do not provide the specified functionality.

“The first 4000 degens to dump NFTs will be able to mint a rat and join the Secret Rat Society, a one of a kind alpha group. You will also be able to stake your rats to earn Junkcoin passively.”

No such system is implemented.

The code should not violate the requirements provided by the Customer.

Path: ./contracts/JKD.sol

Recommendation: Implement described functionality or update documentation.

Status: Mitigated (The first feature will be handled off-chain. The second one has been removed from the documentation.)

■■■ High

1. Highly Permissive Role Access - Requirements Violation

The documentation states that “In 365 days after launch, the Great Burn will commence and all NFTs left in the Junkyard smart contract will be burned, lost forever!”. In the contract, there is no time restriction for the owner to do the great burn (contract self destruct).

In this case, all not-released chain native coins will be transferred to the contract owner instead of splitting between initial payees.

Path: ./contracts/JKD.sol

Recommendation: The code should not violate the requirements provided by the Customer.

Status: Mitigated (The litepaper explains: “The burning mechanism will be integrated into the smart contracts at a later stage, and the exact date will be determined by the Secret Rat Society via DAO.”)

2. Requirements Violation

According to the documentation, users can fish from 1 to 60 NFTs. Contracts have predefined prices only for attempts with quantities equal to 1, 3, 5, 7, 15, 40 and 60. Owner of the contract can change the max amount of fished NFTs to any desired number.

The code should not violate the requirements provided by the Customer.

Path: ./contracts/JKD.sol : function fishing()

Recommendation: Update data in the contract and limit max NFTs able to fish in one time or update documentation.

Status: Mitigated (we offer 1, 3, 7, 15, 40 and 60 and not all numbers from 1-60)

3. Data Consistency

It is possible to register new tokens with already existing hashes or Id in functions `registerNewToken` and `registerCollection`. Additionally, it is possible to use duplicate `_registrationTx` and `_id` in the function `registerNewFishingAttempt`.

This can bring many practical problems at the application level, making the system untrustworthy to the outside world.

Path: ./contracts/JKDManager.sol : functions registerNewToken(), registerCollection(), registerNewFishingAttempt()

Recommendation: Add additional sanity checks for function parameters.

Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

4. **Data Consistency**

   The same token from toWinTokens[] can be pushed several times to request.wonTokens[].

   Path: ./contracts/JKDManager.sol : function fulfillRandomWords()

   Recommendation: Remove the token from the source array when it becomes part of claimable tokens. Then, put it back in the source array if the user chooses another token.

   Status: Fixed (8a3fba537baa793ce2cc6f6768284d85e6b0c9e7)

5. **Highly Permissive Role Access**

   The JKDManager contract owner can change the VRFCoordinator address. In case of updating VRFCoordinator, it cannot be validated if the response came from a trusted source.

   Path: ./contracts/JKD.sol

   Recommendation: Remove the possibility of changing VRFCoordinator.

   Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

6. **Data Consistency - Denial Of Service**

   The function *fulfillRandomWords* pushes a certain amount of tokens in the *wonTokens* array. These tokens are taken from the *availableTokens* array. However, if there are not enough tokens available, the function will run an infinite loop until reaching an out of Gas exception.

   Path: ./contracts/JunkYardManager.sol : function fulfillRandomWords()

   Recommendation: Either verify that there are enough tokens available or exit the loop when there is no more token available.

   Status: Fixed (7f018433e5e8370913abcabda2fcb89559388231)

Medium

1. **Best Practice Violation - Unchecked Transfer**

   It is considered following best practices to avoid unclear situations and prevent common attack vectors.

   Using the transferFrom function does not check if the contract recipients are aware of the ERC721 protocol to prevent tokens from being forever locked.

   Path: ./contracts/JKD.sol : function claimLoot()

   Recommendation: Follow common best practices, use *safeTransferFrom* function.

Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

2. **Missing Event for Critical Value Updation**

   Critical state changes should emit events for tracking things off-chain.

   The functions *setPrice*, *pause*, *unpause* from JKD.sol contract and functions *registerNewToken*, *setSubscriptionId*, *setCoordinator*, *setKeyhash*, *setRequestConfirmation*, *registerCollection* does not emit events on change of important values.

   This can lead to inability of users to subscribe events and check what is going on with the project.

   Paths: ./contracts/JKD.sol : functions setPrice(), pause(), unpause()

   ./contracts/JKDManager.sol : functions registerNewToken(), setSubscriptionId(), setCoordinator(), setKeyhash(), setRequestConfirmation(), registerCollection()

   Recommendation: Emit events on critical state changes.

   Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

3. **Missing Event for Critical Value Updation**

   Critical state changes should emit events for tracking things off-chain.

   The functions *setManagerAddress*, *setManagerChain*, *setStorageAddress*, *setStorageChain* from Junkyard.sol contract, functions *setJunkyardAddress*, *setJunkyardChain*, *setStorageAddress*, *setStorageChain*, *setPoopAddress*, *setPoopChain* from JunkyardManager.sol, functions *setManagerChain*, *setManagerAddress* from JunkyardStorage.sol contract does not emit events on change of important values.

   This can lead to inability of users to subscribe events and check what is going on with the project.

   Paths: ./contracts/Junkyard.sol : functions setManagerAddress(), setManagerChain(), setStorageAddress(), setStorageChain()

   ./contracts/JunkyardManager.sol : functions setJunkyardAddress(), setJunkyardChain(), setStorageAddress(), setStorageChain(), setPoopAddress(), setPoopChain()

   ./contracts/JunkyardStorage.sol : functions setManagerChain(), setManagerAddress()

   Recommendation: Emit events on critical state changes.

   Status: Fixed (8a3fba537baa793ce2cc6f6768284d85e6b0c9e7)

Low

1. **State Variables Default Visibility**

Variables *toWinTokens*, *subscriptionId*, *callbackGasLimit*, *requestConfirmations*, *COORDINATOR* and *keyHash* visibility are not specified. Specifying state variables' visibility helps to catch incorrect assumptions about who can access the variable.

This makes the contract`s code quality and readability higher.

Path: ./contracts/JKDManager.sol

Recommendation: Specify variables as public, internal, or private. Explicitly define visibility for all state variables.

Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

2. **Variables that Should Be Declared Constant**

State variables that do not change their value (*callbackGasLimit*) should be declared constant to save Gas.

Path: ./contracts/JKDManager.sol

Recommendation: Declare the above-mentioned variable as constants.

Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

3. **Floating Pragma**

The project uses floating pragma ^0.8.4.

Paths: ./contracts/JKD.sol

./contracts/JKDManager.sol

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

Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

4. **Style Guide Violation**

In JKD.sol and JKDManager.sol, the events should be placed after the state variables and before the constructor.

In JKDManager.sol, the external function should be before the public one. The internal function should be after public ones. The public view functions should be placed after the public non-view functions.

Event names should be in CapWords.

COORDINATOR should be in mixedCase.

Paths: ./contracts/JKD.sol

./contracts/JKDManager.sol

Recommendation: Follow the official Solidity guidelines.

Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

5. **Commented Code Parts**

In the contract JKDManager.sol line 38 is a commented part of code.

Path: ./contracts/JKDManager.sol

Recommendation: Remove commented parts of code.

Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

6. **Functions that Can Be Declared External**

"public" functions that are never called by the contract should be declared "external" to save Gas.

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

Paths: ./contracts/JKD.sol : functions fishing(), claimLoot(), setPrice(), pause(), unpause(), theGreatBurn()

./contracts/JKDManager.sol : functions registerNewToken(), registerNewFishingAttempt(), claim(), getToken(), getRequestStatus(), getTotalTokenToWin(), setSubscriptionId(), setCoordinator(), setKeyhash(), setRequestConfirmation(), registerCollection()

Recommendation: Use the external attribute for functions never called from the contract.

Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

7. **Variable Shadowing**

Constructor arguments *_payees, _shares* shadows corresponding variables from PaymentSplitter.sol

This makes the contract`s code quality and readability higher.

Path: ./contracts/JKD.sol

Recommendation: Rename related variables/arguments.

Status: Fixed (a9576b1371919c2f24ead0266f002151d11c269e)

8. **State Variables Can Be Declared Immutable**

In the contract JunkyardManager.sol, variables COORDINATOR and subscriptionId values are set in the constructor. These variables can be declared immutable.

This will lower the Gas taxes.

Path: ./contracts/JunkyardManager.sol

Recommendation: Declare mentioned variables as immutable.

Status: Fixed (8a3fba537baa793ce2cc6f6768284d85e6b0c9e7)

9. **Unindexed Events**

Having indexed parameters in the events makes it easier to search for these events using indexed parameters as filters.

Paths: ./contracts/Junkyard.sol : NewFishingEntry, NewClaim

./contracts/JunkyardManager.sol : TokenRegistred, CollectionRegistred

./contracts/JunkyardStorage.sol : TokenSended

Recommendation: Use the “indexed” keyword to the event parameters.

Status: Fixed (8a3fba537baa793ce2cc6f6768284d85e6b0c9e7)

10. Missing Zero Address Validation

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Paths: ./contracts/Junkyard.sol : constructor()

./contracts/JunkyardManager.sol : constructor(), registerNewToken(), registerNewFishingAttempt(), registerCollection()

./contracts/JunkyardStorage.sol : constructor()

Recommendation: Implement zero address checks.

Status: Fixed (8a3fba537baa793ce2cc6f6768284d85e6b0c9e7)

11. Functions that Can Be Declared External

“public” functions that are never called by the contract should be declared “external” to save Gas.

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

Path: ./contracts/JunkyardStorage.sol : setManagerChain(), setManagerAddress()

Recommendation: Use the external attribute for functions never called from the contract.

Status: Fixed (8a3fba537baa793ce2cc6f6768284d85e6b0c9e7)

12. Unused ERROR

Unused libraries/imports/functions/arguments/errors should be removed from the contracts. This will help lower the Gas cost.

Path: ./contracts/Junkyard.sol : error NotEnoughValueForGas

Recommendation: Remove the redundant error or use it.

Status: Fixed (8a3fba537baa793ce2cc6f6768284d85e6b0c9e7)

13. Redundant Use

The use of the variable “nonce” is not needed. randomWords[i] can be incremented directly.

Creating this unnecessary variable will increase Gas cost.

Path: ./contracts/JunkyardManager.sol : uint256 nonce

Recommendation: Remove the unnecessary variable.

Status: Fixed (7f018433e5e8370913abcabda2fcb89559388231)

14. Inconsistent Data

The event TokenRegistred is not indexed by the correct parameter.

Path: ./contracts/JunkyardManager.sol : TokenRegistred

Recommendation: Put the “indexed” keyword for the address.

Status: Fixed (7f018433e5e8370913abcabda2fcb89559388231)

15. Inconsistent Data

The first parameter of the event is supposed to be “value name” and the second one “new value”. The new value is assigned to the two parameters.

Path: ./contracts/JunkyardStorage.sol : functions setManagerChain(), setManagerAddress()

Recommendation: Assign the value name for the first parameter.

Status: Fixed (7f018433e5e8370913abcabda2fcb89559388231)

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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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, Consultant cannot guarantee the explicit security of the audited smart contracts.