Kingdom Raids

Smart Contract Audit Report

SYSFIXED

10 Sep 2022

TABLE OF CONTENTS

AUDITED DETAILS

Audited Project

Project name	Token ticker	Blockchain
Kingdom Raids	KRS	Binance Smart Chain

Addresses

Contract address	0x37b53894e7429f794b56f22a32e1695567ee9913
Contract deployer address	0x7Fab1e1ed769B4a15D8EFC811375e96A026487A6

Project Website

https://kingdomraids.io/

Codebase

https://bscscan.com/address/0x37b53894e7429f794b56f22a32e1695567ee9913#code

SUMMARY

Kingdom Raids is a blockchain-based strategy Role-playing game (RPG) developed by a leading gaming studio, TAG Labs. The game takes place in a fictional kingdom, “Dood Kingdom,” a land of mystery and adventure with over 40 heroes in the magical world, each possessing unique abilities.

Contract Summary

Documentation Quality

Kingdom Raids provides a very good documentation with standard of solidity base code.

- The technical description is provided clearly and structured and also dont have any high risk issue.

Code Quality

The Overall quality of the basecode is standard.

- Standard solidity basecode and rules are already followed by Kingdom Raids with the discovery of several low issues.

Test Coverage

Test coverage of the project is 100% (Through Codebase)

Audit Findings Summary

- SWC-101 | It is recommended to use vetted safe math libraries for arithmetic operations consistently on lines 594, 640, 594 and 640.
- SWC-103 | Pragma statements can be allowed to float when a contract is intended on lines 8, 93, 110, 301, 386, 576 and 610.

CONCLUSION

We have audited the Kingdom Raids project released on September 2022 to discover issues and identify potential security vulnerabilities in Kingdom Raids Project. This process is used to find technical issues and security loopholes which might be found in the smart contract.

The security audit report provides satisfactory results with low-risk issues.

The Kingdom Raids smart contract code issues do not pose a considerable risk. The writing of the contract is close to the standard of writing contracts in general. The low-risk issues are some arithmetic operation issues, and a floating pragma is set. The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

AUDIT RESULT

Article	Category	Description	Result
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	PASS
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	ISSUE FOUND
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	PASS
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	ISSUE FOUND
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	PASS
Unprotected Ether Withdrawal	SWC-105	Due to missing or insufficient access controls, malicious parties can withdraw from the contract.	PASS
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	PASS
Reentrancy	SWC-107	Check effect interaction pattern should be followed if the code performs recursive call.	PASS
Uninitialized Storage Pointer	SWC-109	Uninitialized local storage variables can point to unexpected storage locations in the contract.	PASS
Assert Violation	SWC-110 SWC-123	Properly functioning code should never reach a failing assert statement.	PASS
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	PASS
Delegate call to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	PASS

DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	PASS
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	PASS
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	PASS
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	PASS
Signature Unique ID	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	PASS
Incorrect Constructor Name	SWC-118	Constructors are special functions that are called only once during the contract creation.	PASS
Shadowing State Variable	SWC-119	State variables should not be shadowed.	PASS
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	PASS
Write to Arbitrary Storage Location	SWC-124	The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations.	PASS
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. The rule of thumb is to inherit contracts from more /general/ to more /specific/.	PASS
Insufficient Gas Griefing	SWC-126	Insufficient gas griefing attacks can be performed on contracts which accept data and use it in a sub-call on another contract.	PASS
Arbitrary Jump Function	SWC-127	As Solidity doesnt support pointer arithmetics, it is impossible to change such variable to an arbitrary value.	PASS

Typographical Error	SWC-129	A typographical error can occur for example when the intent of a defined operation is to sum a number to a variable.	PASS
Override control character	SWC-130	Malicious actors can use the Right-To-Left-Override unicode character to force RTL text rendering and confuse users as to the real intent of a contract.	PASS
Unused variables	SWC-131 SWC-135	Unused variables are allowed in Solidity and they do not pose a direct security issue.	PASS
Unexpected Ether balance	SWC-132	Contracts can behave erroneously when they strictly assume a specific Ether balance.	PASS
Hash Collisions Variable	SWC-133	Using abi.encodePacked() with multiple variable length arguments can, in certain situations, lead to a hash collision.	PASS
Hardcoded gas amount	SWC-134	The transfer() and send() functions forward a fixed amount of 2300 gas.	PASS
Unencrypted Private Data	SWC-136	It is a common misconception that private type variables cannot be read.	PASS

SMART CONTRACT ANALYSIS

Started	Friday Sep 09 2022 23:54:57 GMT+0000 (Coordinated Universal Time)
Finished	Saturday Sep 10 2022 04:34:48 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Main Source File	TransparentUpgradeableProxy.sol

Detected Issues

ID	Title	Severity	Status
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	ARITHMETIC OPERATION "-" DISCOVERED	low	acknowledged
SWC-101	COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED	low	acknowledged
SWC-101	COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 594

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- TransparentUpgradeableProxy.sol

Locations

```
593   constructor(address _logic, bytes memory _data) payable {
594   assert(_IMPLEMENTATION_SLOT ==
bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1));
595   _upgradeToAndCall(_logic, _data, false);
596   }
597
598
```

SWC-101 | ARITHMETIC OPERATION "-" DISCOVERED

LINE 640

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- TransparentUpgradeableProxy.sol

Locations

```
639   constructor(address _logic, address admin_, bytes memory _data) payable
ERC1967Proxy(_logic, _data) {
640   assert(_ADMIN_SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1));
641   _changeAdmin(admin_);
642   }
643
644
```

SWC-101 | COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED

LINE 594

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- TransparentUpgradeableProxy.sol

Locations

```
593   constructor(address _logic, bytes memory _data) payable {
594   assert(_IMPLEMENTATION_SLOT ==
bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1));
595   _upgradeToAndCall(_logic, _data, false);
596   }
597
598
```

SWC-101 | COMPILER-REWRITABLE "<UINT> - 1" DISCOVERED

LINE 640

low SEVERITY

This plugin produces issues to support false positive discovery within mythril.

Source File

- TransparentUpgradeableProxy.sol

Locations

```
639   constructor(address _logic, address admin_, bytes memory _data) payable
ERC1967Proxy(_logic, _data) {
640   assert(_ADMIN_SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1));
641   _changeAdmin(admin_);
642   }
643
644
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 8

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- TransparentUpgradeableProxy.sol

Locations

```
7
8   pragma solidity ^0.8.0;
9
10   /**
11   * @dev This abstract contract provides a fallback function that delegates all calls to another contract using the EVM
12
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 93

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- TransparentUpgradeableProxy.sol

Locations

```
92
93    pragma solidity ^0.8.0;
94
95    /**
96    * @dev This is the interface that {BeaconProxy} expects of its beacon.
97
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 110

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- TransparentUpgradeableProxy.sol

Locations

```
109
110   pragma solidity ^0.8.0;
111
112   /**
113   * @dev Collection of functions related to the address type
114
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 301

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- TransparentUpgradeableProxy.sol

Locations

```
300
301   pragma solidity ^0.8.0;
302
303   /**
304   * @dev Library for reading and writing primitive types to specific storage slots.
305
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 386

low SEVERITY

The current pragma Solidity directive is ""^0.8.2"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- TransparentUpgradeableProxy.sol

Locations

```
385
386   pragma solidity ^0.8.2;
387
388
389
390
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 576

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- TransparentUpgradeableProxy.sol

Locations

```
575
576   pragma solidity ^0.8.0;
577
578
579
580
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 610

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- TransparentUpgradeableProxy.sol

Locations

```
609
610   pragma solidity ^0.8.0;
611
612
613   /**
614
```

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to, or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Sysfixed's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Sysfixed to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model, or legal compliance.

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Sysfixed and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Sysfixed) owe no duty of care.

ABOUT US

Sysfixed is a blockchain security certification organization established in 2021 with the objective to provide smart contract security services and verify their correctness in blockchain-based protocols. Sysfixed automatically scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts. Sysfixed a comprehensive range of analysis techniques—including static analysis, dynamic analysis, and symbolic execution—can accurately detect security vulnerabilities to provide an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, Sysfixed can be utilized in all phases of your project's lifecycle. Our team of security experts is dedicated to the research and improvement of our tools and techniques used to fortify your code.