RD AUDITORS

CoinToss Smart Contract, Code Review and Security Analysis Report

Customer: BetSwirl
Prepared on: 3rd March 2022
Platform: Polygon
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report of BetSwirl
Platform	Polygon / Solidity
File 1	Address.sol
MD5 hash	7974C57E068E12E328DE062C9A0B1459
SHA256 hash	D859226F723DE18F916393E5B995A7EF26376C03CD63FC7B683ACBA2C1BA0FEA
File 2	CoinToss.sol
MD5 hash	E2B1A837A3E88CCAC75F451A6CE6DA1E
SHA256 hash	DE863FA3D0822B1785622745E2F05A583982D9577C6DEC1FE34467C40A22A100
File 3	Context.sol
MD5 hash	27A4F3776DAB827CA4ACCE0CDAB957A4
SHA256 hash	8DD98BD63B738695C5CD76A01A662B280774A926848B4E368DD61EB5267EEBD7
File 4	Game.sol

MD5 hash	5162C1415163071C03A3E19F88F0728F
SHA256 hash	EE20587B11BE5C23ACC0D562ED800EB97CAE8A1B212FC46E52B487E35E28B1C0
File 5	IBank.sol
MD5 hash	717ACC0EEEFB1B37CB9E82BC9077D031
SHA256 hash	E230AE0FF790C4587D2C87DE911772F22BD9D12A0D7C362EEA67BF6032EB7786
File 6	IERC20.sol
MD5 hash	12C28F0A0E7A5741C8B87EE17EE3B0C8
SHA256 hash	A164C224F2F502F16E8A9F5B2FC627C55B62A09BEC97499CC01565A053E8939D
File 7	IReferral.sol
MD5 hash	8C95AA60A706F1A6D293BD47B6BCC27D
SHA256 hash	A69B0EF2447E66337046E860CF010D51764F6F0FB0CDD64F9F8F6B0CB22AD2DF
File 8	Multicall.sol
MD5 hash	EFA3E708ABD79CF62303252DB0C13FBB

SHA256 hash	C48BA29836D94EDB54C8B43212666DD6358EDBC7C49CD2B57C138B6140A95E33
File 9	Ownable.sol
MD5 hash	71A6FB0C7388468AB3B8ADD972075615
SHA256 hash	AA5A29CF5EAFA333485FB0C992C1D2D5E9621664569E844B17804E0823D2F788
File 10	Pausable.sol
MD5 hash	A5F3C370F5FDCBB28CCF1F715EA12C1C
SHA256 hash	B315EF9BCA3C569354726C60557227F3BBBC95D3A02F9AF26A4D307870290F3F
File 11	SafeERC20.sol
MD5 hash	DB9B29F27F76C82A75C887131C3B4F00
SHA256 hash	7952192D4EF9BBF66B52C2EFF169BBE6D7CE72446552944B24DD406ADA6BD032
File 12	VRFConsumerBaseV2.sol
MD5 hash	937CCAD064C5C540DB1C79F987988449

SHA256 hash	6D6BAE10F8ECBBAE94B40C7C3A1A7340948BDBA2C89B2EB4524BC7466D3DDCEF
File 13	VRFCoordinatorV2Interface.sol
MD5 hash	7B8A0B1B5FCB015909AC42205CC7E1F4
SHA256 hash	461290AB144F71D4AAC183FB67CA8D50C8B6521B1312E0AB0C8FB93D97301C6B
Date	03/02/2022

Introduction

RD Auditors (Consultant) were contracted by CoinToss (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contracts and its code review conducted between 26th February - 3rd March 2022.

This contract consists of thirteen files.

Project Scope

The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, the customer's solidity smart contract is **well-secured.**

Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found the following;

Total Issues	0
Critical	0
High	0
Medium	0
Low	0
Very Low	0

Code Quality

Please note that within this report safeMath, IERC20, EnumerableSet Math, SafeERC20, ReentrancyGuard, Pausable, Address, ownable are taken from the popular OpenZeppelin library.

The libraries within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times by other contracts.

The BetSwirl team has provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is well commented. Commenting can provide rich documentation for functions, return variables and more. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the BetSwirl code as a link:

https://polygonscan.com/address/0x67CF8C56c09d747dB83a94d5828C4dBcB13487F9

The hash of that file is mentioned in the table. As mentioned above, It's recommended to write comments in the smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

CoinToss

File And Function Level Report

File: CoinToss.sol

Contract: CoinToss

Import: Game

Inherit: Game

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	setHouseEdge	write	Passed	All Passed	No Issue	Passed
2	wager	write	Passed	All Passed	No Issue	Passed
3	fulfillRandomWords	internal	Passed	All Passed	No Issue	Passed
4	getLastUserUnresolvedBets	read	Passed	All Passed	No Issue	Passed
5	getPayout	read	Passed	All Passed	No Issue	Passed

File: Game.sol

Contract: CRAOwner

Import: Ownable, Multicall,

Pausable, VRFConsumerBaseV2

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_newBet	write	Passed	All Passed	No Issue	Passed
2	_resolveBet	write	Passed	All Passed	No Issue	Passed
3	_getFees	read	Passed	All Passed	No Issue	Passed
4	setTokenMinBetAmount	write	Passed	All Passed	No Issue	Passed
5	_setHouseEdge	internal	Passed	All Passed	No Issue	Passed
6	refundBet	write	Passed	All Passed	No Issue	Passed
7	_getLastUserUnresolvedBets	read	Passed	All Passed	No Issue	Passed
8	pause	write	Passed	All Passed	No Issue	Passed
9	setChainlinkConfig	write	Passed	All Passed	No Issue	Passed
10	inCaseTokensGetStuck	write	Passed	All Passed	No Issue	Passed
11	setBank	write	Passed	All Passed	No Issue	Passed
12	setReferralProgram	write	Passed	All Passed	No Issue	Passed

File: Multicall.sol

Contract: Multicall

Import: Address

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	multicall	read	Passed	All Passed	No Issue	Passed

File: Pausable.sol

Contract: Pausable

Import: Context

Inherit: Context

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	paused	read	Passed	All Passed	No Issue	Passed
2	_pause	write	Passed	All Passed	No Issue	Passed
3	_unpause	write	Passed	All Passed	No Issue	Passed

File: VRFConsumerBasev2.sol

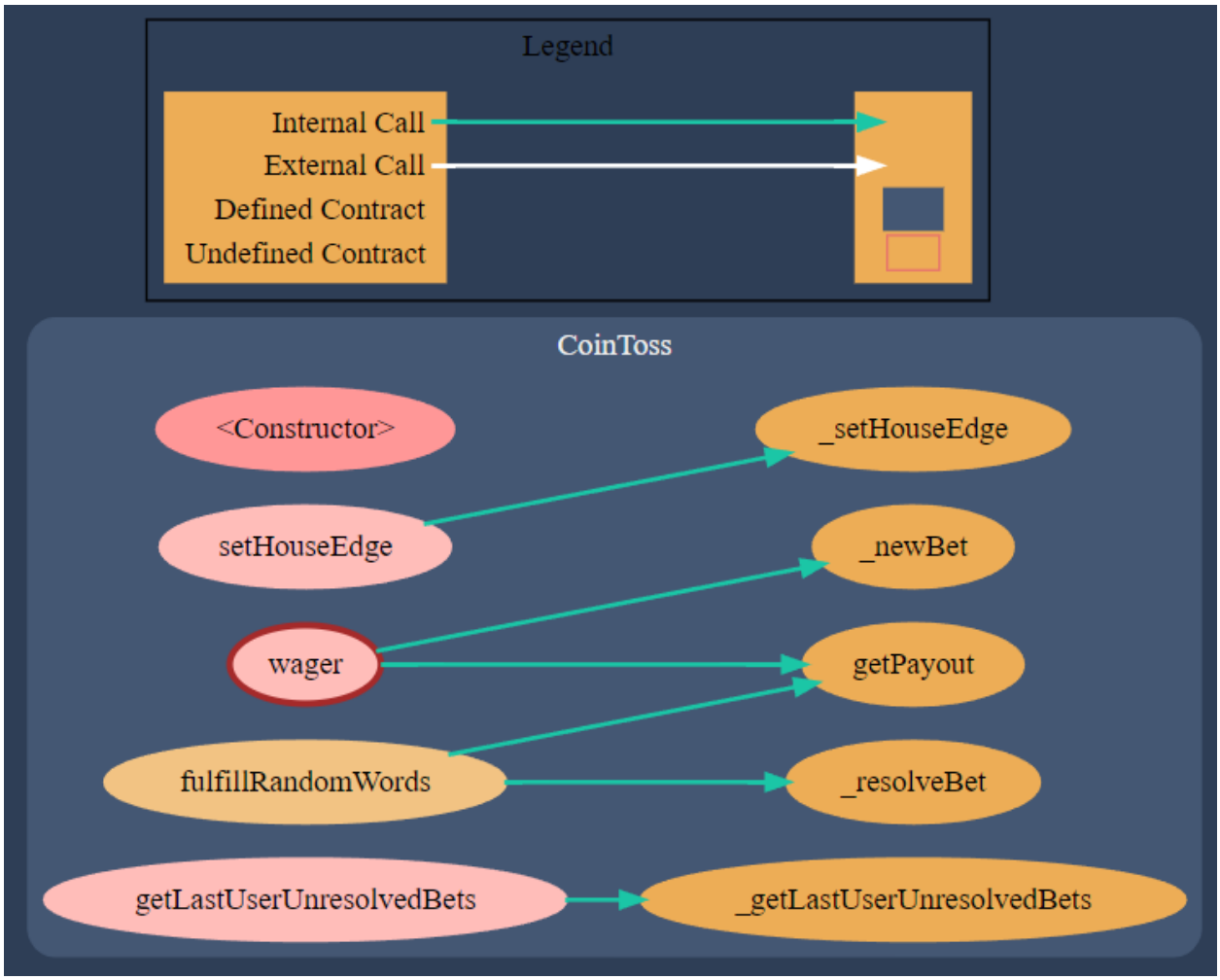
Contract: VRFConsumerBasev2

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	fulfillRandomWords	internal	Passed	All Passed	No Issue	Passed
2	rawFulfillRandomWords	write	Passed	All Passed	No Issue	Passed

Code Flow Diagram



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
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 lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical:

No critical severity vulnerabilities were found.

High:

No high severity vulnerabilities were found.

Medium:

No medium severity vulnerabilities were found.

Low:

No low severity vulnerabilities were found.

Very Low:

No very low severity vulnerabilities were found.

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically, so it is ready to go for production.

We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is now “well secured”.

Note For Contract Users

Owner has full control over the smart contract. Thus, technical auditing does not guarantee the project's ethical side.

Please do your due diligence before investing. Our audit report is never an investment advice.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors Disclaimer

The smart contracts given for audit have been analysed in accordance with the best industry practices at the date of this report, in relation to: 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).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. 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 security of smart contracts.

Technical Disclaimer

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

Email: info@rdauditors.com

Website: www.rdauditors.com