



# BetSwirl Dice Smart Contract, Code Review and Security Analysis Report

---

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

**[rdauditors.com](https://rdauditors.com)**

---

# Table of Contents

<b>Disclaimer</b>	<b>2</b>
<b>Document</b>	<b>3</b>
<b>Introduction</b>	<b>7</b>
<b>Executive Summary</b>	<b>9</b>
<b>Code Quality</b>	<b>10</b>
<b>Documentation</b>	<b>11</b>
<b>AS-IS Overview</b>	<b>13</b>
<b>Observation</b>	<b>18</b>
<b>Conclusion</b>	<b>18</b>
<b>Code Flow Diagram</b>	<b>22</b>
<b>Severity Definitions</b>	<b>25</b>
<b>Audit Findings</b>	<b>26</b>
<b>Discussion</b>	<b>27</b>
<b>Conclusion</b>	<b>29</b>
<b>Note For Contract Users</b>	<b>30</b>
<b>Disclaimers</b>	<b>33</b>

---

## 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	IReferral.sol
MD5 hash	BCFCBCE233F715E7BD44D2454E4CC7F3
SHA256 hash	DCFBDE8166192FC7E26DCD65050E36488EBC750D4F2F3722A51366FC5F13F6E8
File 2	Dice.sol
MD5 hash	2AE79F479138B27FCC4231CB868D62B4
SHA256 hash	28F9D8A4CD630491E36F9A5DB22E1C249D362F29BCD1112D3A46B9B692552662
File 3	Game.sol
MD5 hash	5977938271CB4E67EAC26139A127F41F

SHA256 hash	03D53766FAB06C964D41ECC7B0CDCE7 CC6D9614D9956B48B575B94EF554523C 5
File 4	Context.sol
MD5 hash	7093B039D79750162610C51CD75DFCE8
SHA256 hash	6EE66D1E4693EC63D29393CC2C835947 98475AD0EEABCB0951A35780EF003113
File 5	IBank.sol
MD5 hash	D0EEAE82138FD2084A4A474711E8D6CA
SHA256 hash	A6A1BCC34070ABB98D3F2BFDA6D0B9 3DCBC58B41D354F15BD7AB90F147F495 F7
File 6	IERC20.sol
MD5 hash	3B16C5EA13B835E58AE0AFE036BF7E1F
SHA256 hash	B2AF3659E0D81F48F80F845845309365 B3C3B407F90AAA13E0D26217DB73A727
File 7	Ownable.sol

MD5 hash	86D00D592D33469229ED5E5C8668A4B F
SHA256 hash	405EBD4EC9DC6C0094FCE62214712864 081E43B7FE831D9584C534B843CC8461
File 8	Pausable.sol
MD5 hash	D5BF6FDA7EA9F759E356F34868B31C54
SHA256 hash	55A3662D60825043ACCBAA18899B080 05A2CA0B6585FB495453F63C4B564E89 B
File 9	SafeERC20.sol
MD5 hash	52546A054058393497A7CA9992ED3991
SHA256 hash	29BA4766454D69501875EEAED7FA6B07 31F4E8808EC99399F82FA39E8F54DC6A
File 10	Multicall.sol
MD5 hash	52BD1C311E4C0A6BB910C00496AAF069
SHA256 hash	83EB6A379272BD729DCE253DC5426AD 64296EDB8C5236EACF3170F5A999473E 4

File 11	VRFConsumerBaseV2.sol
MD5 hash	861DFB3C01D33E61DCF2526E808413F0
SHA256 hash	4D272724490EC032285A313F8566A8C9 0E213C36797410F20635A169E41E1103
File 12	VRFCoordinatorV2Interface.sol
MD5 hash	D00338177C0122E4529C610626EA2A22
SHA256 hash	4E88AE1F4A3D8469BDA1F16AF40F60CF 770E362B228092AE82884921752B70B5
File 13	Address.sol
MD5 hash	B0DDD9373ECE24F1B38E3853981BF598
SHA256 hash	7F97CA5E12463C83F4EEF63445553D61F 1D17674CF3EDF5AF4A3F29F87DBF407
Date	03/03/2022

---

## Introduction

RD Auditors (Consultant) were contracted by BetSwirl (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 13 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**.

You are Here

 **Insecure**     **Poorly Secured**     **Secure**     **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, 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 find a link that, within this report Address, Pausable, ownable, Multicall and SafeERC20 etc taken from the popular open source.

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 helped 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 the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

---

## Documentation

We were given the Dice source code as a Github link:

<https://polygonscan.com/address/0xC38842415049F265f6714154BD3Fc120c554b54C>

The hash of that file is mentioned in the table. As mentioned above, It's well commented 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

### BetSwirl (Dice) Smart Contract

#### File and Function Level Report

Contract: Dice  
Inherit: Game  
Import: Ownable, Multicall  
Observation: Passed  
Test Report: Passed  
Score: Passed  
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	SetHouseEdgeAndMinCap	OnlyOwner	Passed	All Passed	No Issue	Passed
2	Wager	external	Passed	All Passed	No Issue	Passed
3	FulfillRandomness	internal	Passed	All Passed	No Issue	Passed
4	getLastuserUnresolvedBets	external	Passed	All Passed	No Issue	Passed
5	getPayout	read	Passed	All Passed	No Issue	Passed

Contract: Game

Import: Address, Ownable, Multicall, Pausable, SafeERC20,  
VRFCConsumerBaseV2, VRFCo-ordinatorV2Interface, IBank, IReferral

Inherit: Ownable, Pausable, Multicall, VRFCConsumerBaseV2

Observation: Passed

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_SetHouseEdge	onlyOwner	Passed	All Passed	No Issue	Passed
2	_newBet	internal	Passed	All Passed	No Issue	Passed
3	_resolveBet	internal	Passed	All Passed	No Issue	Passed
4	_getLastUserUn resolvedBets	internal	Passed	All Passed	No Issue	Passed
5	_getFees	internal	Passed	All Passed	No Issue	Passed
6	SetTokenMinBet Amount	onlyowner	Passed	All Passed	No Issue	Passed
7	Pause	onlyowner	passed	All Passed	No Issue	Passed
8	SetChainlinkCo nfig	onlyowner	Passed	All Passed	No Issue	Passed
9	InCaseTokenGet Stuck	onlyowner	Passed	All Passed	No Issue	Passed
10	refundBet	external	Passed	All Passed	No Issue	Passed
11	SetBank	onlyowner	Passed	All Passed	No Issue	Passed
12	SetReferralProgr am	external	passed	All Passed	No Issue	Passed

Interface: IReferral  
Observation: Passed  
Test Report: Passed  
Score: Passed  
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	PayReferral	external	Passed	All Passed	No Issue	Passed
2	hasReferrer	external	Passed	All Passed	No Issue	Passed

Contract: Context  
Observation: Passed  
Test Report: Passed  
Score: Passed  
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_msgSender	internal	Passed	All Passed	No Issue	Passed
2	_msgData	internal	Passed	All Passed	No Issue	Passed



Interface: IBank.sol  
Observation: Passed  
Test Report: Passed  
Score: Passed  
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	isAllowedToken	read	Passed	All Passed	No Issue	Passed
2	payout	external	Passed	All Passed	No Issue	Passed
3	CashIn	external	Passed	All Passed	No Issue	Passed
4	addReferrer	external	Passed	All Passed	No Issue	Passed
5	UpdateReferrerActivity	external	Passed	All Passed	No Issue	Passed
6	hasReferrer	external	Passed	All Passed	No Issue	Passed
7	getMaxSetAmount	external	Passed	All Passed	No Issue	Passed
8	getTokenMinBetTxValue	external	Passed	All Passed	No Issue	Passed

Interface IERC20  
Observation: Passed  
Test Report: Passed  
Score: Passed  
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	totalSupply	read	Passed	All Passed	No Issue	Passed
2	balanceOf	read	Passed	All Passed	No Issue	Passed
3	transfer	external	Passed	All Passed	No Issue	Passed
4	allowance	read	Passed	All Passed	No Issue	Passed
5	approve	external	Passed	All Passed	No Issue	Passed
6	transferFrom	external	Passed	All Passed	No Issue	Passed

Contract: Ownable

Inherit: Context

Observation: Passed

Test Report: Passed

Score: Passed

Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Owner	read	Passed	All Passed	No Issue	Passed
2	renounceOwnership	onlyowner	Passed	All Passed	No Issue	Passed
3	transferOwnership	onlyowner	Passed	All Passed	No Issue	Passed
4	_transferOwnership	internal	Passed	All Passed	No Issue	Passed

Contract: Pausable  
Inherit: Context  
Observation: Passed  
Test Report: Passed  
Score: Passed  
Conclusion: Passed

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

Abstract: VRFConsumerBaseV2  
Observation: Passed  
Test Report: Passed  
Score: Passed  
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	fulfillRandomwords	internal	Passed	All Passed	No Issue	Passed
2	rawfulfillRandomword	external	Passed	All Passed	No Issue	Passed

Library: SafeERC20  
Import: IERC20, Address  
Observation: Passed  
Test Report: Passed  
Score: Passed  
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	SafeTransfer	internal	Passed	All Passed	No Issue	Passed
2	SafeTransferFrom	internal	Passed	All Passed	No Issue	Passed
3	SafeApprove	internal	Passed	All Passed	No Issue	Passed
4	SafeIncreaseAllowance	internal	Passed	All Passed	No Issue	Passed
5	SafeDecreaseAllowance	internal	Passed	All Passed	No Issue	Passed
6	_CallOptionalReturn	Private	Passed	All Passed	No Issue	Passed

Library: VRFCoordinatorV2Interface  
Observation: Passed  
Test Report: Passed  
Score: Passed  
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getRequestConfig	external	Passed	All Passed	No Issue	Passed
2	requestRandomWords	external	Passed	All Passed	No Issue	Passed
3	CreateSubscription	external	Passed	All Passed	No Issue	Passed
4	getSubscription	external	Passed	All Passed	No Issue	Passed
5	requestSubscriptionOwnerTransfer	external	Passed	All Passed	No Issue	Passed
6	addConsumer	external	Passed	All Passed	No Issue	Passed
7	removeConsumer	external	Passed	All Passed	No Issue	Passed
8	CancelSubscription	external	Passed	All Passed	No Issue	Passed

Abstract: Multicall

Import: Address  
Observation: Passed

Test Report: Passed

Score: Passed

Conclusion: Passed

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

Library: Address

Observation: Passed

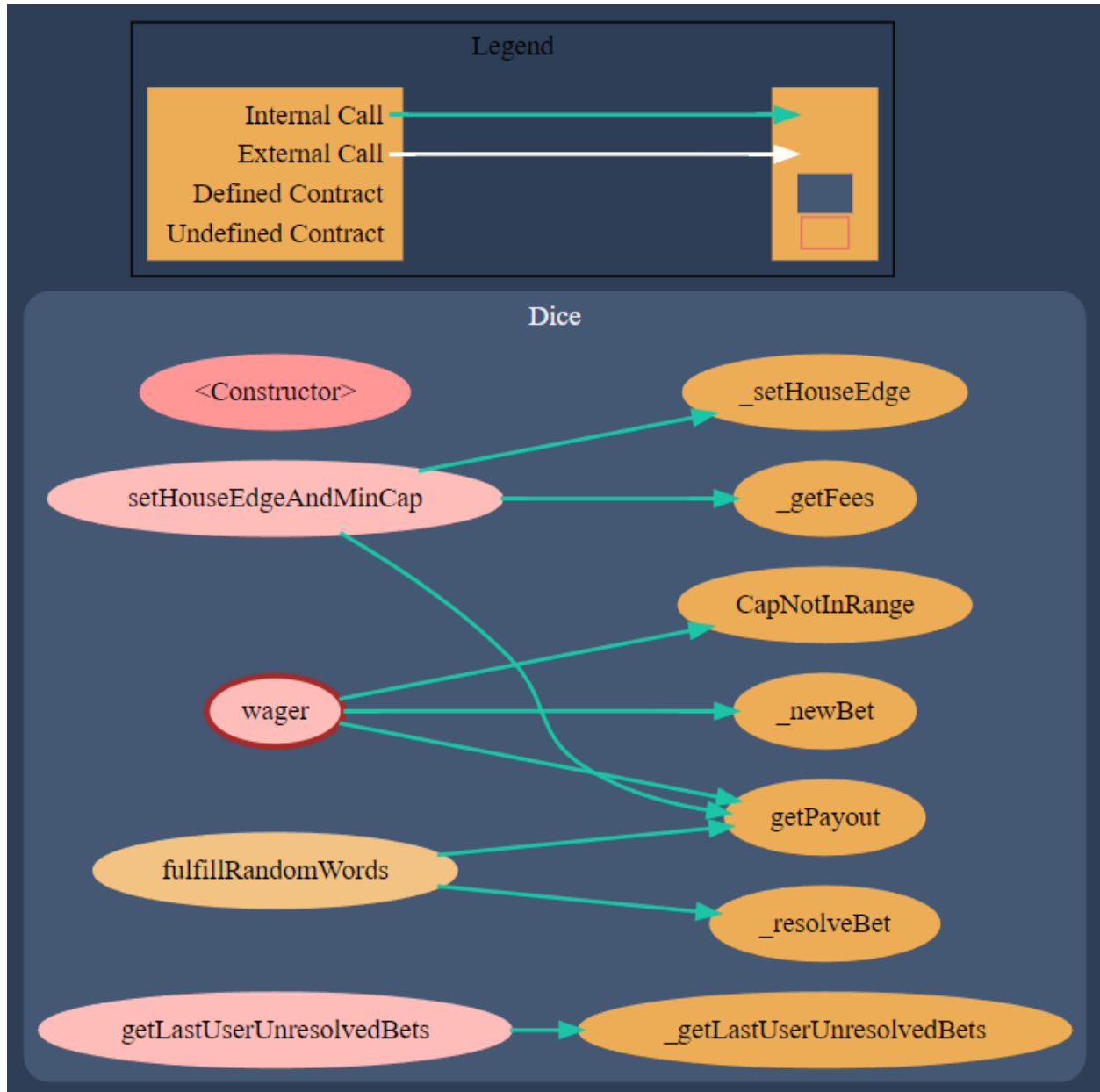
Test Report: Passed

Score: Passed

Conclusion: Passed

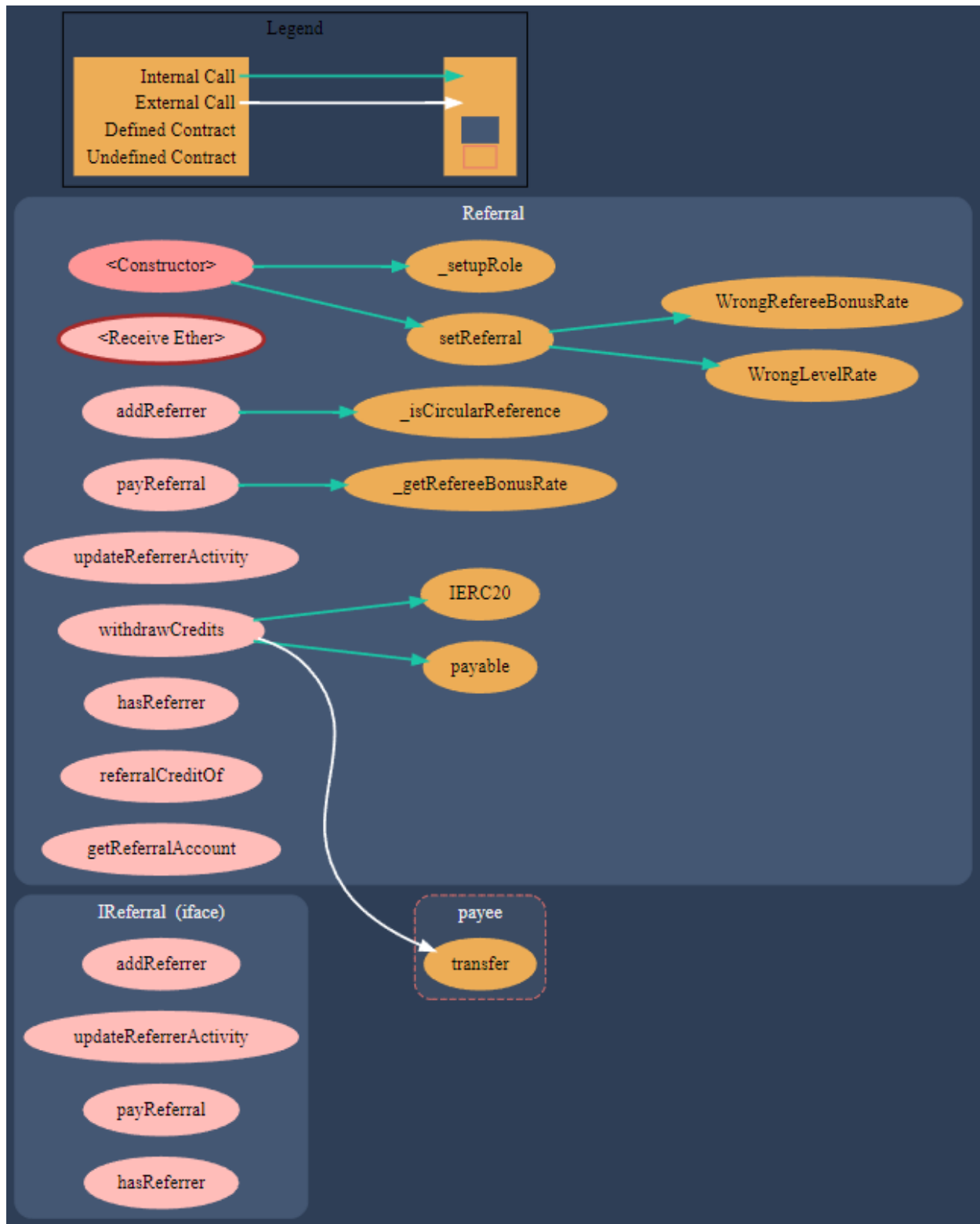
Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	isContract	internal	Passed	All Passed	No Issue	Passed
2	sendValue	internal	Passed	All Passed	No Issue	Passed
3	functionCall	internal	Passed	All Passed	No Issue	Passed
4	functionCall	internal	Passed	All Passed	No Issue	Passed
5	functionCallWithValue	internal	Passed	All Passed	No Issue	Passed
6	functionCallWithValue	internal	Passed	All Passed	No Issue	Passed
7	functionStaticCall	internal	Passed	All Passed	No Issue	Passed
8	functionStaticCall	internal	Passed	All Passed	No Issue	Passed
9	functionDelegateCall	internal	Passed	All Passed	No Issue	Passed
10	functionDelegateCall	internal	Passed	All Passed	No Issue	Passed
11	VerifyCallResult	internal	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.

## Discussion

### File: Dice.sol

1) Loop may fail if unresolvedBets.length is too high.

```
177     function getLastUserUnresolvedBets(address user, uint256 dataLength)
178     external
179     view
180     returns (DiceBet[] memory)
181     {
182         Bet[] memory unresolvedBets = _getLastUserUnresolvedBets(
183             user,
184             dataLength
185         );
186         DiceBet[] memory unresolvedDiceBets = new DiceBet[](
187             unresolvedBets.length
188         );
189         for (uint256 i; i < unresolvedBets.length; i++) {
190             unresolvedDiceBets[i] = DiceBet(
191                 unresolvedBets[i],
192                 diceBets[unresolvedBets[i].id]
193             );
194         }
195         return unresolvedDiceBets;
196     }
```

### File: Game.sol

2) Please be aware that the function `inCaseTokensGetStuck()` at line number #447 and #449 will allow the owner to withdraw all the funds and tokens from the contract.

```
442     function inCaseTokensGetStuck(address token, uint256 amount)
443     external
444     onlyOwner
445     {
446         if (token == address(0)) {
447             Address.sendValue(payable(msg.sender), amount);
448         } else {
449             IERC20(token).safeTransfer(msg.sender, amount);
450         }
451     }
```

3) In line 293 it is very essential to pass Gas amount with the call like below example

```
function callValueEther(uint _amount) public payable {  
    require(address(receiverAdr).call.value(_amount).gas(35000)());  
}
```

```
265 function _resolveBet(  
266     Bet storage bet,  
267     bool wins,  
268     uint256 payout  
269 ) internal returns (uint256) {  
270     address payable user = bet.user;  
271     if (bet.resolved == true || user == address(0)) {  
272         revert NotPendingBet(bet.id);  
273     }  
274     address token = bet.token;  
275     uint256 betAmount = bet.amount;  
276     bool isGasToken = bet.token == address(0);  
277  
278     bet.resolved = true;  
279  
280     // Check for the result  
281     if (wins) {  
282         uint256 profit = payout - betAmount;  
283         uint256 betAmountFee = _getFees(token, betAmount);  
284         uint256 profitFee = _getFees(token, profit);  
285         uint256 fee = betAmountFee + profitFee;  
286  
287         payout -= fee;  
288  
289         uint256 betAmountPayout = betAmount - betAmountFee;  
290         uint256 profitPayout = profit - profitFee;  
291         // Transfer the bet amount from the contract  
292         if (isGasToken) {  
293             (bool success, ) = user.call{value: betAmountPayout}("");  
294             if (!success) {
```

---

## 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

There are several owner only functions. Those can be called by the owner's wallet only. So, if the owner's wallet is compromised, then it carries the risk of the contract becoming vulnerable.

The owner can drain all the funds and tokens by calling #inCaseTokensGetStuck() function.

```
442     function inCaseTokensGetStuck(address token, uint256 amount)
443     |
444     |   external
445     |   |
446     |   |   onlyOwner
447     |   |   {
448     |   |   |   if (token == address(0)) {
449     |   |   |   |   Address.sendValue(payable(msg.sender), amount);
450     |   |   |   |   } else {
451     |   |   |   |   IERC20(token).safeTransfer(msg.sender, amount);
452     |   |   |   |   }
453     |   |   |   }
454     |   |   }
```

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

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](mailto:info@rdauditors.com)

Website: [www.rdauditors.com](http://www.rdauditors.com)

