RD AUDITORS

BetSwirl Token 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	BetsToken.sol
MD5 hash	BD75FE4856899880251759E48237EC23
SHA256 hash	D0D025706442F89FF1852C387D4F0DB6E1E6DA074BD36C07C4A36AD8656E26EA
File 2	Multicall.sol
MD5 hash	0667D75648E4237DAF12AFABF458AE18
SHA256 hash	464EEB7E7F459B3A90AA77BCBAA45F90ECFD9197ACE55F1DFAF853E76CB72E9D
File 3	Address.sol
MD5 hash	9535D34FF7265770FEBEE12BD6748118

SHA256 hash	51BC5D75E334916802BB89AF4F137C052E25F90A169B87E932E4CA14D06D7743
File 4	ERC20.sol
MD5 hash	C593A270E0D1669F5F98E97A3FCD7222
SHA256 hash	847C7CCA5FCD1C2AA5AB8047B6D8CFF160C23FE905DC059C7D4DE7E53D349FBD
File 5	IERC20.sol
MD5 hash	91B1F5D488F122784A31767FDE096831
SHA256 hash	891DCE62E2F334BDBCE74319DDCC65BEA60EBB38AEA89430B9E8B6A4C2D1347F
File 6	IERC20Metadata.sol
MD5 hash	C246F74893BA341C6408302DAE51323B
SHA256 hash	E792EC419549394322BE7B0489819DDA41D1FDC4C8082401EBE5556C1B083641
File 7	Context.sol

MD5 hash	7AFE318D316CA6D6D94ADD38564AF31C
SHA256 hash	42DAF2750AF0B626E39DBB21F497975A B16616A652B13397FFDCFCA477F96471
File 8	Ownable.sol
MD5 hash	2E548C50A460987FC68A628B3A9F31B2
SHA256 hash	5D710EA67FFD6DB825F4A6E6B141A175F 72AD804FA5D05CD3F93F735866F38D5
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 8 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, ERC20, ownable and SafeERC20, 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 source code of BetSwirl token contract as a link:

https://polygonscan.com/token/0x9246a5f10a79a5a939b0c2a75a3ad196aafdb43b

The hash of that file is mentioned in the table. As mentioned above, It's recommended to write comments on 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 Token Smart Contract

File And Function Level Report

File:	BetsToken
Contract:	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 WithValue	internal	Passed	All Passed	No Issue	Passed
5	functionCallw ithValue	internal	Passed	All Passed	No Issue	Passed
6	functionStati cCall	internal	Passed	All Passed	No Issue	Passed
7	functionStati cCall	internal	Passed	All Passed	No Issue	Passed
8	functionDele gateCall	internal	Passed	All Passed	No Issue	Passed
9	functionDele gateCall	internal	Passed	All Passed	No Issue	Passed
10	VerifyCallRes ult	internal	Passed	All Passed	No Issue	Passed

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

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getOwner	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

Contract:	ERC20
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	Passed

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

2	symbol	read	Passed	All Passed	No Issue	Passed
3	decimals	read	Passed	All Passed	No Issue	Passed
4	totalSupply	read	Passed	All Passed	No Issue	Passed
5	balanceOf	read	Passed	All Passed	No Issue	Passed
6	transfer	write	Passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	approve	write	Passed	All Passed	No Issue	Passed
9	transferFrom	write	Passed	All Passed	No Issue	Passed
10	increaseAllowance	write	Passed	All Passed	No Issue	Passed
11	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
12	transfer	internal	Passed	All Passed	No Issue	Passed
13	_mint	internal	Passed	All Passed	No Issue	Passed
14	burn	internal	Passed	All Passed	No Issue	Passed
15	_approve	internal	Passed	All Passed	No Issue	Passed
16	_beforeTokenTransfer	internal	Passed	All Passed	No Issue	Passed
17	_afterTokenTransfer	internal	Passed	All Passed	No Issue	Passed

Interface:	IERC20
Import:	IERC20, IERCMetadata, context
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	Passed

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

Interface:	IERC20Metadata
Import:	IERC20
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	external	Passed	All Passed	No Issue	Passed
2	symbol	external	Passed	All Passed	No Issue	Passed
3	decimals	external	Passed	All Passed	No Issue	Passed

Contract:	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

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	_setOwner	write	Passed	All Passed	No Issue	Passed

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

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.

RD
AUDITORS

Email: info@rdauditors.com

Website: www.rdauditors.com