RD AUDITORS

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Wall Street Bets
Prepared on: 16/04/2021
Platform: Binance Smart Chain
Language: Solidity

info@rdauditors.com

TABLE OF CONTENTS

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 DECISION OF CUSTOMER.

info@rdauditors.com

Document

Name	Smart Contract Code Review and Security Analysis Report for Wall Street Bets
Platform	BSC / Solidity
File	AdminUpgradeabilityProxy.sol
MD5 hash	7100F76981C029EAAE14F35AD0EBF38B
SHA256 hash	0FDAD1001AD10F87E35F350AAB64464B0C37F587C5A20A280F458CFDFFBDB3E7
Date	16/04/2021

info@rdauditors.com

Introduction

RD Auditors (Consultant) was contracted by Wall Street Bets (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contracts and its code review conducted between April 13, 2021 – April 16, 2021.

This contract consists of a single file.

Project Scope

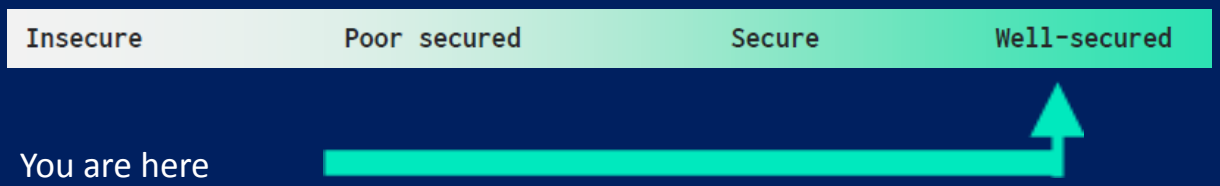
The scope of the project is a smart contract.

We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but 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, 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 0 critical, 0 high, 0 medium, 0 low and 0 very low level issues.

Note: The admin rights to the proxy contract has been burnt, and also transferred to the dead address (details below). This has turned the token into a non-upgradeable proxy.

https://bscscan.com/tx/0x64dd5fe65591822eb6fc8ffc66f344c2cd645002713f73f73d0ab19ce7d27b74#eventlog

Code Quality

AdminUpgradeAbilityProxy consists of a single smart contract file. This single file smart contract also contains openzeppelinupgradesAddress from the popular open source.

The library in the AdminUpgradeAbilityProxy is part of its 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 a specific address and its properties / methods can be reused many times by other contracts in the AdminUpgradeAbilityProxy.

WSB has **not** 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 provides rich documentation for functions, return variables and more and also helps auditors to quick cover the flow behind code logic. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given AdminUpgradeAbilityProxy in the form of the link:

https://bscscan.com/address/0x22168882276e5d5e1da694343b41dd7726eeb288#code

The hash of that file is mentioned in the table. As mentioned, 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 library is used in this smart contract infrastructure. That was based on well known industry standard open source projects. And even core code blocks are written well and systematically.

AS-IS Overview

AdminUpgradeAbilityProxy Overview

AdminUpgradeAbilityProxy provides the facility of changing the contract and administrators via proxy method.

File And Function Level Report

Contract: openzeppelinUpgradesOwnable
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	write	Passed	All Passed	No Issue	Passed
3	transferOwnership	write	Passed	All Passed	No Issue	Passed
4	_transferOwnership	write	Passed	All Passed	No Issue	Passed

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

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_implementation	read	Passed	All Passed	No Issue	Passed
2	_delegaate	write	Passed	All Passed	No Issue	Passed
3	_willFallback	write	Passed	All Passed	No Issue	Passed
4	_fallback	write	Passed	All Passed	No Issue	Passed

Contract: BaseUpgradeabilityProxy
inherit: Proxy
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_implementation	read	Passed	All Passed	No Issue	Passed
2	_upgradeTo	write	Passed	All Passed	No Issue	Passed
3	_setImplementation	write	Passed	All Passed	No Issue	Passed

Contract: BaseAdminUpgradeabilityProxy
inherit: BaseUpgradeabilityproxy
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	admin	write	Passed	All Passed	No Issue	Passed
2	implementation	write	Passed	All Passed	No Issue	Passed
3	changeAdmin	write	Passed	All Passed	No Issue	Passed
4	UpgradeTo	write	Passed	All Passed	No Issue	Passed
5	UpgradeToAndCall	write	Passed	All Passed	No Issue	Passed
6	_admin	read	Passed	All Passed	No Issue	Passed
7	_setadmin	write	Passed	All Passed	No Issue	Passed
8	_willFallback	write	Passed	All Passed	No Issue	Passed

Contract: AdminUpgradeabilityProxy
inherit: BaseUpgradeabilityProxy, upgradeabilityproxy
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Contract: ProxyAdmin
inherit: openzeppelinUpgradesOwnable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getProxyImplementation	read	Passed	All Passed	No Issue	Passed
2	getProxyAdmin	read	Passed	All Passed	No Issue	Passed
3	changeProxyAdmin	write	Passed	All Passed	No Issue	Passed
4	Upgrade	write	Passed	All Passed	No Issue	Passed
5	UpgradeAndCall	write	Passed	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lossed tokens etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution e.g. public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; but does not lead to lossed tokens.
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that generally don´t have a significant impact on execution.
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can’t affect smart contract execution and can be ignored.

Audit Findings

Critical

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.

info@rdauditors.com

Conclusion

We checked the whole contract in line with given transaction link https://bscscan.com/tx/0x64dd5fe65591822eb6fc8ffc66f344c2cd645002713f73f73d0ab19ce7d27b74#eventlog where admin was burnt to address 0x0000....00001 instead of address 0. As the address 0 was restricted by the code (due to this contract being already in production), there was no way to burn to address 0. Additionally, the level of difficulty for finding a private key for address 0 is almost the same as address 1, so essentially this transaction can be treated as admin burnt with no issue/option of retrieving it.

We also checked other parts of the code pertaining to this aspect and the findings are as follows:

1. All admin functions are now inaccessible to any individual.
2. No vulnerabilities were found.

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically.

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such extensive smart contract protocol, we provide no such guarantee of future outcomes. 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”.

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 analyze 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 scrutinized 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 analyzed 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, so the audit makes no statements or warranties on 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 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

info@rdauditors.com