RD
AUDITORS

OVR SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer:	OVR
Prepared on:	04/10/2021
Platform:	BSC
Language:	Solidity

.

TABLE OF CONTENTS

Document

Name	**Smart Contract Code Review and Security Analysis Report for OVR**
Platform	BSC / Solidity
File 1	Initializable.sol
MD5 hash	891ECBD9283CB44253055711B55354D6
SHA256 hash	A2AAEBC80B5E5278C849336C4E9498343EF6ADD8B21D43FB3F4C87D5A5175BA5
File 2	OVRToken.sol
MD5 hash	A0C039B3CB2FC96765CCD5258F855A36
SHA256 hash	60527CC3FECC2A072F10E2BC38B6D9E25912DF71F775D1050C5E0896D3E2E339
File 3	Ownable.sol
MD5 hash	6406B9994519172D8D13B32A1024D11D
SHA256 hash	DD7BA50F5D3221E38ED15283800F66941C278341A647298AFE01986894EEDD17
File 4	ReentrancyGuard.sol
MD5 hash	5B84E9291AFD5573D60C667B09C25AAA
SHA256 hash	F79B41F9736D76F5B627A3F8F09B1C378CE59D690E85686D42D7752F77B6D963
File 5	StakingV2
MD5 hash	93BE06F952987C86A105F1629983ABCF
SHA256 hash	FC333D4E41BDEA37189D6D128B57E77BD4B16F6E9B153D00FD72A152D4898A81
File 6	Context.sol

MD5 hash	34B249F23F865FFB147ABDED2AE9266A
SHA256 hash	1978DA26CB892AB5593A18F31E43E69223FFCAA4262F9488ACDDACE56A11741B
Date	04/10/2021

Introduction

RD Auditors (Consultant) were contracted by OVR (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 8 - 13 September and
1 - 4 October 2021.

We have checked 6 files of this contract, which are in the above table.

Project Scope

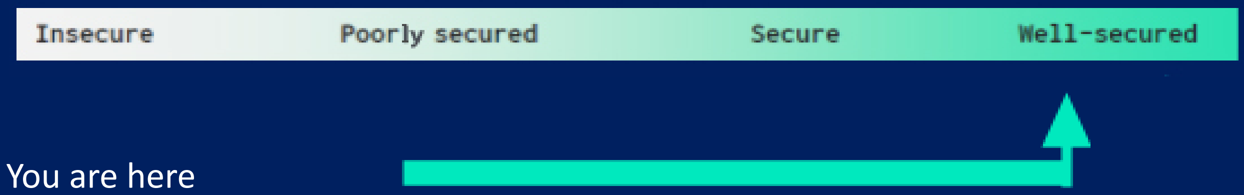
The scope of the project is a smart contract.

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, 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.

Code Quality

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 OVR team has also conducted unit tests using scripts provided through the same github link which fortify functionality and security of the contract, which also helped us 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 OVRStakingV2 contract as a github link

https://github.com/OVR-Platform/stakingV2

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

StakingV2

File And Function Level Report

File: StakingV2 .sol

Contract:	StakingV2
Import:	Console, IERC20Mintable, IERC20, SafeMath, Address, SafeERC20, ABDKMath64*64, ABDKMathQuad.sol, Initializable, sacrifice, ReentrancyGuard, Context, Ownable
Inherit:	Ownable, ReentrancyGuard, OVRStaking
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	PauseContract	write	Passed	All Passed	No Issue	Passed
2	PauseDepositAndLockupExtensions	write	Passed	All Passed	No Issue	Passed
3	initializeStaking	write	Passed	All Passed	No Issue	Passed
4	setLiquidityProviderAddress	write	Passed	All Passed	No Issue	Passed
5	deposit	write	Passed	All Passed	No Issue	Passed
6	isLockUpPeriodExpired	read	Passed	All Passed	No Issue	Passed
7	pow	read	Passed	All Passed	No Issue	Passed
8	compound	read	Passed	All Passed	No Issue	Passed
9	calcRewards	read	Passed	All Passed	No Issue	Passed
10	getCurrentBalance	read	Passed	All Passed	No Issue	Passed
11	LiquidityProviderAddress	read	Passed	All Passed	No Issue	Passed

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

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

Contract:	OVRToken
Import:	Console
Inherit:	ERC20, Interface, SafeMath
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	Passed

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

Contract:	Ownable
Import:	console, context, Initializable
Inherit:	Initializable, context
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	initialize	write	Passed	All Passed	No Issue	Passed
2	Owner	read	Passed	All Passed	No Issue	Passed
3	isOwner	read	Passed	All Passed	No Issue	Passed
4	renounceOwnership	write	Passed	All Passed	No Issue	Passed
5	transferOwnership	write	Passed	All Passed	No Issue	Passed

Contract:	ReentrancyGuard
Import:	Initializable
Inherit:	Initializable
Observation:	Passed
Test Report:	Passed
Score:	Passed
Conclusion:	Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	initialize	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.

Not For User

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

- **PauseContract:** Within the smart contract, the owner can stop, pause deposits, withdraw and lockup extensions.
- **PauseDepositAndLockUpExtension:** The owner can pause deposit and lockup extension.
- **SetLiquidityProviderAddress:** Only the owner can set the address for the liquidity provider's reward.

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.

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such an 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 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