RD AUDITORS

MERCOR FINANCE SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Mercor Finance
Prepared on: 29/04/2021
Platform: Binance Smart Chain
Language: Solidity

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.

Document

Name	Smart Contract Code Review and Security Analysis Report for Mercor Finance
Platform	BSC / Solidity
File 1	MercorPrivateSale.sol
MD5 hash	9DFBBBB0ED24C16BDF5867313C25375D
SHA256 hash	11F0B809DE484FBC501CB821368BC2A5F91FBD278F8E415BEF386D033F5C57A9
File 2	MercorToken.sol
MD5 hash	8C6CD4F1E072EA707E7435578886052C
SHA256 hash	C316F75BC776124C6C80EED0F940CA1A73F3E5F745688B1241E0EE5D2CC36393
File3	IBEP20.sol
MD5 hash	78B259B06B1912C1B6E181C81C8555F4
SHA256 hash	7DE709BD44686721696E05C4F2E5C8AB638D2FFE627CE67CC920309D854E0F38
Date	28/04/2021

Introduction

RD Auditors (Consultant) was contracted by Mercor Finance (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of the customer`s smart contracts and its code review conducted between 22- 29 April 2021.

This contract consists of three files.

Project Scope

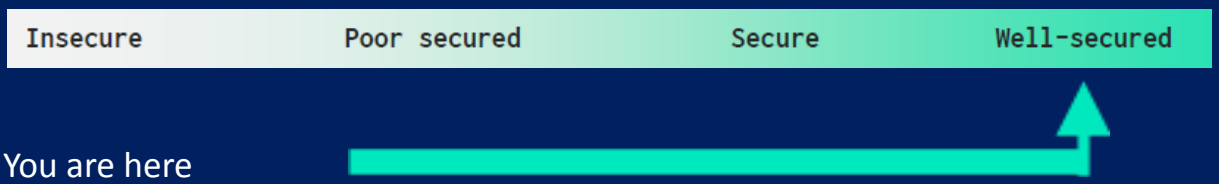
The scope of the project is a smart contract.

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, 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, some medium, 0 low and 0 very low level issues.

UPDATE (29/04/2021): After the code modification we found 0 critical, 0 high, 0 medium, 0 low and 0 very low level issues.

Code Quality

Mercor Finance consists of three smart contract files. This multiple file smart contract also contains SafeMath from the popular open source.

Mercor Finance´s library 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 Mercor´s stack.

Mercor Finance´s team 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

Mercor Finance contract was provided as a file.

The hash of that file is mentioned in the table. As mentioned, it's well commented 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 that are written well and systematically.

AS-IS Overview

Mercor Overview

It is a BEP20 contract for processing phase wise token sales to raise funds.

File And Function Level Report

File: MercorPrivateSale.sol

Contract: MercorPrivateSale
Import: IBEP20.sol, SafeMath.sol
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	deploy	write	Passed	All Passed	No Issue	Passed
2	notifyRewardAmounts	write	Passed	All Passed	No Issue	Passed
3	notifyRewardAmount	write	Passed	All Passed	No Issue	Passed

File: MercorToken.sol

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

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

Contract: Ownable
Import: 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	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: MercorToken
Inherit: Context, IBEP20, ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	getOwner	read	Passed	All Passed	No Issue	Passed
2	decimals	read	Passed	All Passed	No Issue	Passed
3	symbol	read	Passed	All Passed	No Issue	Passed
4	name	read	Passed	All Passed	No Issue	Passed
5	totalSupply	read	Passed	All Passed	No Issue	Passed
6	balanceOf	read	Passed	All Passed	No Issue	Passed
7	transfer	write	Passed	All Passed	No Issue	Passed
8	allowance	read	Passed	All Passed	No Issue	Passed
9	approve	write	Passed	All Passed	No Issue	Passed
10	transferFrom	write	Passed	All Passed	No Issue	Passed
11	increaseAllowance	write	Passed	All Passed	No Issue	Passed
12	decreaseAllowance	write	Passed	All Passed	No Issue	Passed
13	_transfer	write	Passed	All Passed	No Issue	Passed
14	_approve	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 high severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

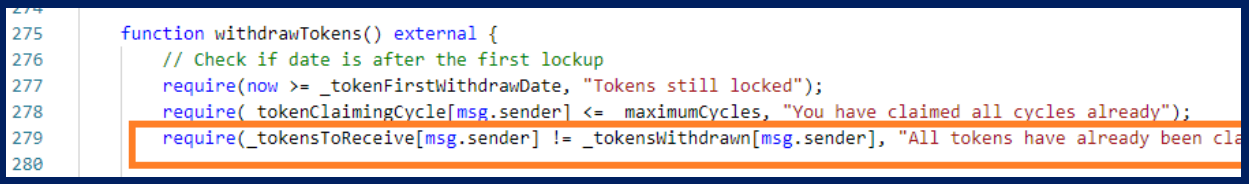
File: MercorPrivateSale.sol

1. Line 271 requires the use of safeMath to avoid overflow. This will also be applied to line 194.

```
266     function unlockRemainingTokens()
267     external
268     onlyOwner
269     {
270         require(now >= _tokenUnlockDate, "Tokens still locked");
271         uint256 balance = _token.balanceOf(address(this))- _tokensSold + _totalTokensWithdrawn;
272         require(_token.transfer(msg.sender, balance), "Token transfer failed");
273     }
274
```

UPDATE (29/04/2021): Following on from our suggestions, shortcomings were removed and thus does not have any vulnerabilities.

2. For enhanced safety and to avoid any minor fractional variation, line 279 should be ">=" instead of "!=".



UPDATE (29/04/2021): Following on from our suggestions the code function "withdrawToken" has been modified and thus does not have any vulnerabilities.

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.

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

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 blockchain platforms. 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