RD AUDITORS

MARSSWAP SMART CONTRACT, CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: MarsSwap
Prepared on: 21 May 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 of MarsSwap
Platform	BSC / Solidity
File 1	GovernorAlpha.sol
MD5 hash	C4C966D260EFE727A255A99E4032AD74
SHA256 hash	D53166F0C7D8DB30C7D9613FD261FF0F9CFCF215781D58FB33CDF627CD2A5FE2
File 2	MarsToken.sol
MD5 hash	E5EF721EFE3C7E1F62595A9A4E9C502A
SHA256 hash	4EF60DFD8FD5B45AE0C1503EC900CB8085352DBE0CF6C411F9B7E7762A41CDAC
File 3	MasterPlanet.sol
MD5 hash	D3FA07A21CF47C59BE29E6BF6045057E
SHA256 hash	7E7A234C3914B7703FCE152EE1843E6A23B58EB5BC35B869B646267BBE0D024C
File 4	Referral.sol
MD5 hash	7737C06B9DBEB5EAD3021C9FC5747B64
SHA256 hash	8071576B97777FD678BBC8FE47643CBF7CBF62DDEF22B28E4D8B0BCABFDEEAD8
File 5	TimeLock.sol
MD5 hash	12F2CC81AB65B6761995585D16B1C615
SHA256 hash	99F4D75ADBB0CE148D03EE46134C54C8B34D54FF58BBA59F9F82CD755B9C3AB9
Date	21/05/2021

Introduction

RD Auditors (Consultant) was contracted by MarsSwap (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
15 - 21 May 2021.

This contract consists of five files.

Project Scope

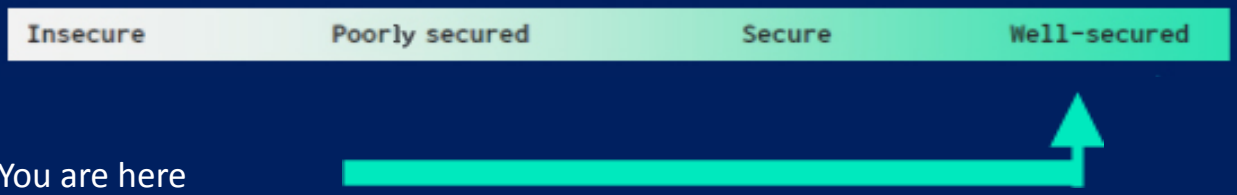
The scope of the project is a smart contract.

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

Code Quality

Please find a link that, within this report, contains SafeMath, ownable, ReentrancyGuard and SafeBIP20 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.

MarsSwap consists of five smart contract files. The MarsSwapFinance team have also conducted unit tests using script 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 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 the Marsswap contract as a github link:
https://github.com/MarsSwapFinance/marsswap-contracts/tree/main/contracts

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 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

MarsSwap

Is a protocol that allows anyone to exchange tokenized assets on Binance Smart Chain

File And Function Level Report

File: GovernorAlpha.sol

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

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	QuorumVotes	read	Passed	All Passed	No Issue	Passed
2	ProposalThreshold	read	Passed	All Passed	No Issue	Passed
3	ProposalMaxoperation	read	Passed	All Passed	No Issue	Passed
4	votingDelay	read	Passed	All Passed	No Issue	Passed
5	VotingPeriod	read	Passed	All Passed	No Issue	Passed
6	Propose	write	Passed	All Passed	No Issue	Passed
7	queue	write	Passed	All Passed	No Issue	Passed
8	_queueOrRevert	write	Passed	All Passed	No Issue	Passed
9	execute	write	Passed	All Passed	No Issue	Passed
10	cancel	write	Passed	All Passed	No Issue	Passed
11	getActions	read	Passed	All Passed	No Issue	Passed
12	getReceipt	read	Passed	All Passed	No Issue	Passed
13	state	read	Passed	All Passed	No Issue	Passed
14	castVote	write	Passed	All Passed	No Issue	Passed
15	castVoteBySig	write	Passed	All Passed	No Issue	Passed
16	_castVote	write	Passed	All Passed	No Issue	Passed
17	add256	read	Passed	All Passed	No Issue	Passed
18	sub256	read	Passed	All Passed	No Issue	Passed
19	getChainId	read	Passed	All Passed	No Issue	Passed

File: MarsToken.sol

Contract: MarsToken
Import: BEP20.sol
Inherit: BEP20
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	mint	write	Passed	All Passed	No Issue	Passed
2	delegates	read	Passed	All Passed	No Issue	Passed
3	delegate	write	Passed	All Passed	No Issue	Passed
4	delegateBySig	write	Passed	All Passed	No Issue	Passed
5	getCurrentVotes	read	Passed	All Passed	No Issue	Passed
6	getPriorVotes	read	Passed	All Passed	No Issue	Passed
7	_delegate	write	Passed	All Passed	No Issue	Passed
8	_moveDelegates	write	Passed	All Passed	No Issue	Passed
9	__writeCheckPoint	write	Passed	All Passed	No Issue	Passed
10	safe32	read	Passed	All Passed	No Issue	Passed
11	getchainId	read	Passed	All Passed	No Issue	Passed

File: MasterPlanet.sol

Contract: MasterPlanet
Import: safeMath, IBEP20, safeBEP20, IReferral
ownable, ReentrancyGuard, MarsToken
Inherit: ownable, ReentrancyGuard
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	PoolLength	read	Passed	All Passed	No Issue	Passed
2	add	write	Passed	All Passed	No Issue	Passed
3	set	write	Passed	All Passed	No Issue	Passed
4	getMultiplier	read	Passed	All Passed	No Issue	Passed
5	pendingMars	read	Passed	All Passed	No Issue	Passed
6	massupdatePools	write	Passed	All Passed	No Issue	Passed
7	updatePool	write	Passed	All Passed	No Issue	Passed
8	deposit	write	Passed	All Passed	No Issue	Passed
9	withdraw	write	Passed	All Passed	No Issue	Passed
10	emergencyWithdraw	write	Passed	All Passed	No Issue	Passed
11	safeMarsTransfer	write	Passed	All Passed	No Issue	Passed
12	dev	write	Passed	All Passed	No Issue	Passed
13	setFeeAddress	write	Passed	All Passed	No Issue	Passed
14	UpdateEmissionRate	write	Passed	All Passed	No Issue	Passed
15	safeMarsUpgrade	write	Passed	All Passed	No Issue	Passed
16	setReferral	write	Passed	All Passed	No Issue	Passed
17	setReferralCommissionRate	write	Passed	All Passed	No Issue	Passed
18	PayReferralCommission	write	Passed	All Passed	No Issue	Passed

File: Referral.sol

Contract: MasterPlanet
Import: IBEP20, safeBEP20, IReferral, ownable
Inherit: IReferral, ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	recordReferral	write	Passed	All Passed	No Issue	Passed
2	getReferrer	read	Passed	All Passed	No Issue	Passed
3	updateOperator	write	Passed	All Passed	No Issue	Passed
4	drainBEP20Token	write	Passed	All Passed	No Issue	Passed

File: TimeLock.sol

Contract: TimeLock
Import: SafeMath, ownable
Inherit: ownable
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	setDelay	write	Passed	All Passed	No Issue	Passed
2	queueTransaction	write	Passed	All Passed	No Issue	Passed
3	cancelTransaction	write	Passed	All Passed	No Issue	Passed
4	executeTransaction	write	Passed	All Passed	No Issue	Passed
5	getBlockTimeStamp	read	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

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low

No very Low severity vulnerabilities were found.

Discussion

1. In file MarsToken contract, MarsToken was written twice. Kindly remove to reduce the file size.

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 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