SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: WhaleSwap Team
Prepared on: 04/03/2021
Platform: Binance Smart Chain
Language: Solidity

audit@etherauthority.io

Table of contents

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO PUBLIC AFTER ISSUES ARE RESOLVED.

Document

Name	Smart Contract Code Review and Security Analysis Report for WhaleSwap
Platform	Binance Smart Chain / Solidity
File name 1	MasterChef.sol
MD5 hash	20e47f3d8d0b9feea4f48a6e1be4e85a
SHA256 hash	3f3f0e6a5ca8413502f2f97a5825d724b16fbfc85c396110e2bd885fe7794511
File name 2	Timelock.sol
MD5 hash	55bd0f0122793b9b0fb18c710d92c6aa
SHA256 hash	297347057c478616f549324331ebaef002dde6b88f8c480932025a63604254aa
File name 3	WhaleToken.sol
MD5 hash	6bb3b65b865da16daa90714554e77331
SHA256 hash	56e132623b2c0cc24324ede4f0a5f01a2d2e40defc6ca4da4e46e48dfbdb09e8
Date	04/03/2021

Introduction

We were contracted by the WhaleSwap team to perform the Security audit of the smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on 04/03/2021.

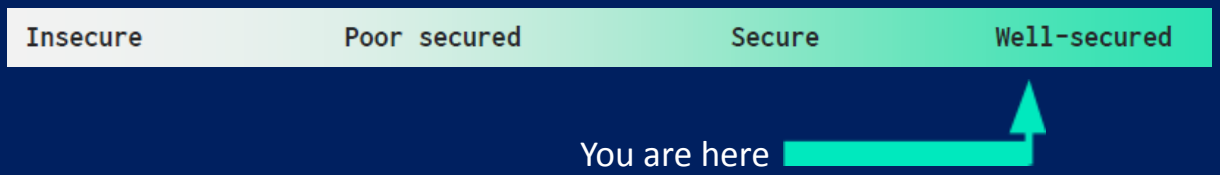
Quick Stats:

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	Assert() misuse	Passed
	High consumption ‘for/while’ loop	Possibility
	High consumption ‘storage’ storage	Passed
	“Out of Gas” Attack	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Executive Summary

According to the assessment, Customer`s solidity smart contract is well secured.



We used various tools like SmartDec, Mythril, Slither and Remix IDE. At the same time this finding is based on critical analysis of the menual audit. All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all found issues can be found in the Audit overview section.

We found 0 high, 1 medium and 1 low and some very low level issues.

Code Quality

WhaleSwap protocol consists of three smart contract files. These smart contracts also contain safeMath, SafeBEP20, IBEP20, Ownable. These are compact and well written contracts.

The libraries in the WhaleSwap protocol are 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 WhaleSwap protocol.

The WhaleSwap 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 **not** 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

As mentioned above, It is not well commented smart contract code, so anyone can not quickly understand the programming flow as well as complex code logic.

We were given a WhaleSwap contract in the form of a file. The hash of that file is mentioned above in the table.

Comments are very helpful in understanding the overall architecture of the protocol. It also provided 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 that are based on well known industry standard open source projects. And even core code blocks are written well and systematically.

AS-IS overview

MasterChef.sol contract overview

MasterChef is a liquidity pool with rewards in Whale token. Following are the main components whose details are explicitly recorded.

(1) Imports:

(a) SafeMath.sol
(b) IBEP20.sol
(c) SafeBEP20.sol
(d) Ownable.sol
(e) WhaleToken.sol

(2) Usages

(a) SafeMath for uint256
(b) SafeBEP20 for IBEP20

(3) Struct

(a) UserInfo - holds all the users information
(b) PoolInfo - holds all the pool information

(4) Events

(a) event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
(b) event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
(c) event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);

(5) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	poolLength	read	Passed	No Issue	Passed
2	add	write	Passed	Need Validation	Passed
3	set	write	Passed	No Issue	Passed
4	getMultiplier	read	Passed	No Issue	Passed
5	pendingEgg	read	Passed	No Issue	Passed
6	massUpdatePools	write	Passed	Gas Limit Possibility	Passed
7	updatePool	write	Passed	No Issue	Passed
8	deposit	write	Passed	No Issue	Passed
9	withdraw	write	Passed	No Issue	Passed
10	emergencyWithdraw	write	Passed	No Issue	Passed
11	safeEggTransfer	Internal	Passed	No Issue	Passed
12	dev	write	Passed	No Issue	Passed
13	setFeeAddress	write	Passed	No Issue	Passed
14	updateEmissionRate	write	Passed	No Issue	Passed

TimeLock.sol contract overview

Timelock contract queues and executes the transactions. Following are the main components whose details are explicitly recorded.

(1) Imports:

(a) SafeMath.sol

(2) Events

(a) event NewAdmin(address indexed newAdmin);
(b) event NewPendingAdmin(address indexed newPendingAdmin);
(c) event NewDelay(uint indexed newDelay);
(d) event CancelTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
(e) event ExecuteTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
(f) event QueueTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);

(3) Functions

Sl.	Function	Type	Observation	Conclusion	Score
1	receive	write	Passed	No Issue	Passed
2	setDelay	write	Passed	No Issue	Passed
3	acceptAdmin	write	Passed	No Issue	Passed
4	setPendingAdmin	write	Passed	No Issue	Passed
5	queueTransaction	write	Passed	No Issue	Passed
6	cancelTransaction	write	Passed	No Issue	Passed
7	executeTransaction	write	Passed	No Issue	Passed
8	getBlockTimestamp	read	Passed	No Issue	Passed

WhaleToken.sol contract overview

WhaleToken is a BEP20 standard token. It has functionality for voting for governance. Following are the main components whose details are explicitly recorded.

(1) Imports:

(a) BEP20.sol

(2) Struct

(a) Checkpoint

(3) Events

(a) event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

(b) event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

(4) Functions

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

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss 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; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have 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

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

(1) Gas limit possibility: massUpdatePools function in MasterChef smart contract might hit the gas limit if the pool size is big.

Low

(1) Validation for _lpToken parameter in the add function in MasterChef smart contract should be added. This is to make sure that is not being setup incorrectly by mistake. Although this is owner only functions, so the chance of raising this issue is low. But still, it's a good idea to put some validations.

Discussion:

(1) Overpowered functions: There are some functions which are authorised persons (set, updateEmissionRate, setFeeAddress) only. And it would be troublesome if its private key would be compromised.

(2) Approve of ERC20 standard: This can be used to front run. From the client side, only use this function to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved).

(3) User latest solidity compiler version.

Conclusion

We were given contract files. And we have used all possible tests based on given objects as files. The contracts are written so systematic, that we did not find any major issues. So it is good to go for production.

Since possible test cases can be unlimited for such extensive smart contract protocol, so 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.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract is "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

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract 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).

Due to the fact that 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 also suggest to conduct a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. 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.


Ether Authority