SecureBlock

Security Audit

Serum

September 2022

Prepared for:
Serum

Prepared by:
SecureBlock
www.secureblock.io

Table of Contents

Security Audit

1 Introduction

This document includes observations and findings during the audit of the smart contract.

a. About SecureBlock

Founded in 2021 by an association of experts in the field of computer security with many years of experience.

Our goal is to simplify and provide a quality security testing service for blockchain projects and technologies. Taking an individual approach and manual review of each project allows us to better understand use case of the applications and find vulnerabilities and problems that standard automated tools will not find.

We believe that openness and trust are one of the key aspects of blockchain technology, which is increasingly finding its purpose in more and more industries. For this reason, our clients have an insight into the state of security testing, a preliminary description of vulnerabilities and the public management of the final report through an application we have developed internally.

b. Purpose of the audit

The purpose of the testing was primarily to find security issues, as well as compliance of the code with best practice and improve code quality.

c. Revision History

Date	Author	Version	Note
16th of Sep 2022	Dalibor T.	v1.0	Initial report
17th of Sep 2022	Luka S.	v1.1	Re-test

2 Executive Summary

a. Results

During the conducted security testing, 0 high-risk, 1 medium and 2 low-risk issues were found.
All issues found during the initial test have been fixed, which is confirmed by this report.

b. Scope

Name	Serum
Language	Solidity
Network	N/A
Source	SerumCoin.sol (SHA1: 22a8b54263f85cb9ac2c82bc0fa41f95917273f3)
Re-test Source	SerumCoin.sol (SHA1: e17f74132381cbbf533cb679d02c0544f99af9e4)

c. Exclusions

Exclusion from testing refers to components and functionalities that we did not have access to during testing, therefore we did not perform test on following:

- *No exclusions*

3 Identified Vulnerabilities

Issue ID	Severity	Title	Status
APP-01	Low	Floating Pragma	Acknowledged
APP-02	Low	Missing Zero Address Check	Fixed
APP-03	Medium	Immutable Variables Cannot be Read From Constructor	Fixed

APP-01 - Floating Pragma

Contract is using floating pragma. Locked pragma ensures that contract does not get accidentally deployed using an unstable compiler version that might introduce bugs.
Status: Acknowledged

APP-02 - Missing Zero Address Check

It has been found that zero address check is missing at multiple locations. Functions *excludeFromMaxTransaction, excludeFromFees, setAutomatedMarketMakerPair, updateSerumWallet* and other functions accepting address type parameter doesn't implement zero address checks.
Status: Fixed

APP-03 - Immutable Variables Cannot be Read From Constructor

It has been found that contract is trying to read immutable variable *uniswapV2Pair* from constructor which causes *TypeError* during compilation.
Status: Fixed

4 Additional Notes

The following notes refer to potential vulnerabilities that are not exploitable in the current environment, but we would like to draw attention as they could cause unexpected behavior in the future if contract gets updated or deployed in different environment.

- *No additional notes*

5 Attack Narrative - Smart Contract

a. Smart Contract Checklist

In order to find vulnerabilities during the test, we go through a checklist that helps us to cover more tests as well as demonstrate to the client which checks were included during testing. In addition to the list below, we check for business logic vulnerabilities that we find on the deployed contract on our local private network so that there are no unexpected consequences for users.

Name	Description
ERC standards	The contract is using ERC standards.
Compiler Version	The compiler version should be specified.
Constructor Mismatch	The constructor syntax is changed with Solidity versions. Need extra attention to make the constructor function right.
Return standard	Following the ERC20 specification, the transfer and approve functions should return a bool value, and a return value code needs to be added.
Address(0) validation	It is recommended to add the verification of require(_to!=address(0)) to effectively avoid unnecessary loss caused by user misuse or unknown errors
Unused Variable	Unused variables should be removed.
Untrusted Libraries	The contract should avoid using untrusted libraries, or the libraries need to be thoroughly audited too.
Event Standard	Define and use Event appropriately
Safe Transfer	Using transfer to send funds instead of send.
Gas consumption	Optimize the code for better gas consumption.
Deprecated uses	Avoid using deprecated functions.
Sanity Checks	Sanity checks when setting key parameters in the system

Name	Description
Integer overflows	Integer overflow or underflow issues.
Reentrancy	Avoid using calls to trade in smart contracts to avoid reentrancy vulnerability.
Transaction Ordering Dependence	Avoid transaction ordering dependence vulnerability.
Tx.origin usage	Avoid using tx.origin for authentication.
Fake recharge	The judgment of the balance and the transfer amount needs to use the “require function”.
Replay	If the contract involves the demands for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks.
External call checks	For external contracts, pull instead of push is preferred.
Weak random	The method of generating random numbers on smart contracts requires more considerations.
Access Control	Well defined access control for functions.
Authentication management	The authentication management is well defined.
Semantic Consistency	Semantics are consistent
Functionality checks	The functionality is well implemented.

6 Vulnerability Remediation

Detailed remediation steps for found issues are available for client over web portal at **https://secureblock.io/dashboard**