SecureBlock

Security Audit

Elgos Company Token

Prepared for:
Elgos
www.elgos.app

Prepared by:
SecureBlock
www.secureblock.io

Table of Contents

1 Introduction

This document includes observations and findings during the audit of the smart contract requested by Elgos. Security testing was provided by senior auditors at SecureBlock.

a. About SecureBlock

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

Our researchers are continuously working on the development of internal tools and knowledge sharing, as well as by holding recognized certificates in the industry such as OSCP, OSCE, OSWE, CEH, CISSP.

Our mission 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, if possible, reduce the gas fee.

2 Executive Summary

a. Results

The conducted testing indicates that the tested smart contract is very safe.

A total of 2 issues were found during the manual review of the smart contract.

The first issue we found was a manual use of arithmetic operators with assignment. Code first inherits the variable's value, perform add/subtract operation and then assigns new variable value. This can be simplified and recude the amount of the code in the contract by using operators with automatic assignment.

The second issue is related to the lack of check before itteration through user-provided array. If the provided array is too long it is possible that transaction runs out of the gas and cause the OUT_OF_GAS exception. As a remediation action we suggest that before iteration, a *require* check is performed to make sure the provided array is not too long.

b. Scope

Contract Name	Elgos Company Token
Network	Binance Smart Chain (BSC) Mainnet
Language	Solidity
Address	0xF37cAc4783a343D6258DdD52AA4e2566C0CbE25a

3 Attack Narative

a. 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

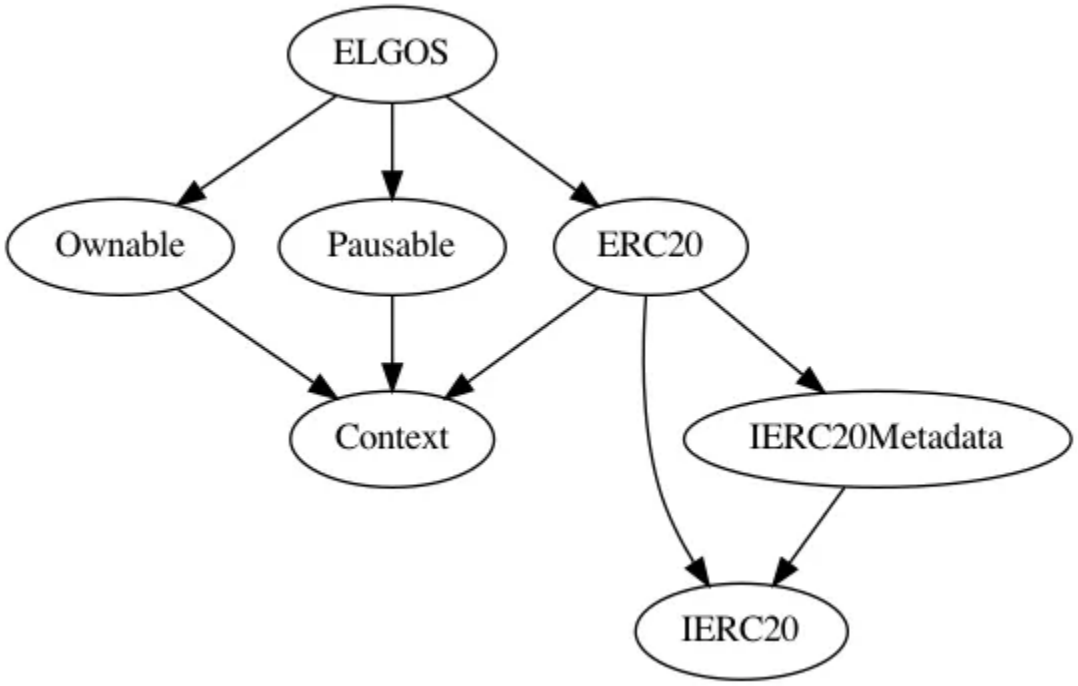
Name	Description
Safe Transfer	Using transfer to send funds instead of send.
Gas consumption	Optimize the code for better gas consumption.
Deprecated uses	Avoid using deprecated functions.
Sanity Checks	Sanity checks when setting key parameters in the system
Integer overflows	Integer overflow or underflow 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.

Name	Description
Authentication management	The authentication management is well defined.
Semantic Consistency	Semantics are consistent
Functionality checks	The functionality is well implemented.

4 Contract Dependences

a. Inheritance

The following content visually describes the inheritance of smart contracts based on the smart contract being tested.



5 Identified Vulnerabilities

Issue ID	Severity	Title	Status
APP-01	Low	Use Arithmetic Operators for Add/Substract with Assignment	Acknowledged
APP-02	Low	Possible OUT_OF_GAS Exception	Acknowledged

APP-01 - Use Arithmetic Operators for Add/Substract with Assignment

It was found that smart contract uses add/subtract and assign operations in a for loop by inheriting the variable and then performing the mathematical operation. We suggest to use arithmetic operators with assignment in Solidity.

Acknowledged: The owner of the project acknowledged the issue. As this issue does not pose a direct threat to the smart contract, there is no need to re-deploy the smart contract.

APP-02 - Possible OUT_OF_GAS Exception

Function *batchTransfer* use for loop to iterate through *_recipient* and *_amountArray* parameters. If the passed array is too long, the contract may return an *OUT_OF_GAS* exception.

Acknowledged: The owner of the project acknowledged the issue. As this issue does not pose a direct threat to the smart contract, there is no need to re-deploy the smart contract.

6 Vulnerability Remediation

Steps on how to remove vulnerabilities and patch reported problems are available for audit client on our portal **https://secureblock.io/login**