SUSTAINABILITY CREATIVE ESHARE (SCCs) SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: SustainabilityCreative.com
Prepared on: 05/05/2021
Platform: Ethereum
Language: Solidity
Audit Type: Standard

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	Code Review and Security Analysis Report for Sustainability Creative eShare (SCCs) Token Smart Contract
Platform	Ethereum / Solidity
File name	SCC_eShare.sol
MD5 hash	B12D973D748A589959E687244F04A1B0
SHA256 hash	ED431ED044A03798583BC90C2E0CD5550FEAD61388F0BD5858A5E91878B5D933
Testnet code URL	https://rinkeby.etherscan.io/address/0x76b22ca486ef3f840f3743f06c84bb4ba930cd7a#code

Introduction

We were contracted by the Sustainability Creative team to perform the Security audit of the SCC_eShare 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 05/05/2021.

Audit type was Standard Audit. Which means one senior auditor performing an audit for 2 days. So, this audit is concluded based on standard audit scope. And because the use case scenarios are unlimited, it is encouraged to perform an Extensive audit (which is performed by 2 or more auditors for about 4 days) to come to a more solid conclusion.

Quick Stats:

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	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	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	N/A
	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 standard audit assessment, Customer`s solidity smart contract is well secured. Again, it is recommended to perform an Extensive audit assessment to bring a more assured conclusion.

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

Code Quality

SCC_eShare consists of one smart contract file. This smart contract also contains inherited Regulated and Ownable smart contracts. These are compact and well written contracts.

Libraries used in the SCC_eShare are part of its logical algorithm. They are smart contracts which contain 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 Sustainability Creative SCCs Token protocol.

The Sustainability Creative 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 can provide rich documentation for functions, return variables and more.

Documentation

As mentioned above, It is a well commented smart contract code. However, this is pretty straightforward ERC20 standard Token smart contract code, with additional features.

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

Other source of the information was obtained from its official website.

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

SCC_eShare.sol contract overview

It is an ERC20 token contract with additional features like acceptETH, regulation, etc.

(1) Libraries

(a) using SafeMath for uint256

(2) Interfaces

(a) ERC20Interface: An interface for ERC20 token standard

(3) Inherits

(a) Context: This provides context like msg.sender and msg.value

(b) Ownable: This manages all ownership functionalities

(c) Regulated: Regulates for the shareholders

(d) AcceptEth: Accepts and refunds ETH from owne

(4) Functions:

Sl.	Function	Type	Observation	Conclusion	Score
1	DissolveBusiness	write	Passed	No Issue	Passed
2	Registershareholder	read	Passed	No Issue	Passed
3	NevadaBlackBook	write	Passed	No Issue	Passed
4	ensureRegulated	read	Passed	No Issue	Passed
5	isRegulated	read	Passed	No Issue	Passed
6	AcceptRefund	write	Passed	No Issue	Passed
7	issue	write	Passed	No Issue	Passed
8	transferOwnership	write	Passed	No Issue	Passed
9	totalSupply	write	Passed	No Issue	Passed
10	balanceOf	write	Passed	No Issue	Passed
11	transfer	write	Passed	No Issue	Passed
12	approve	write	Passed	No Issue	Passed
13	transferFrom	write	Passed	No Issue	Passed
14	allowance	write	Passed	No Issue	Passed
15	transferotherERC20Assest	write	Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token 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

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Best Practices

(1) 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). This should be done from the client side.

(2) Use visibility '*external'* over *'public'.* This is good practice. https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices/19391

Centralization

There are several owner only functions. Those can be called by the owner's wallet only. So, if the owner's wallet is compromised, then it carries the risk of the contract becoming vulnerable to unexpected fate.

- DissolveBusiness : owner can dissolve business any moment
- RegisterShareholder : only Owner can register shareholders
- Accept : to accept ether
- Refund : to refund ether
- Issue : only owner can issue token.
- transferOwnership : Owner can transfer owner to another wallet

Conclusion

We were given contract code. 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