SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Coast Token

AUDIT

SECURITY ASSESSMENT

25. July, 2023

FOR

Introduction

SolidProof.io is a brand of the officially registered company MAKE Network GmbH, based in Germany. We're mainly focused on Blockchain Security such as Smart Contract Audits and KYC verification for project teams.
Solidproof.io assess potential security issues in the smart contracts implementations, review for potential inconsistencies between the code base and the whitepaper/documentation, and provide suggestions for improvement.

Disclaimer

SolidProof.io reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of the security or functionality of the technology we agree to analyze.

Project Overview

Summary

Project Name	COAST
Website	https://0xcoast.com/
About the project	The Coast Stablecoin $CST is a 1-to-1 USD-backed PRC-20 token issued exclusively on #PulseChain. For Coast users, 1 CST is always redeemable for $1 USD in fiat. Use $CST with or without a Coast account to de-risk from #PulseChain assets and lock in appreciation.
Chain	Pulse Chain
Language	Solidity
Codebase Link	https://scan.v4.testnet.pulsechain.com/address/0x7b51434E3911564154251BC4F9D0103fFD99085F/contracts#address-tabs
Unit Tests	Not Provided

Social Medias

Telegram	t.me/coast0x
Twitter	https://twitter.com/0xCoast
Facebook	N/A
Instagram	N/A
Github	N/A
Reddit	N/A
Medium	N/A
Discord	N/A
Youtube	N/A
TikTok	N/A
LinkedIn	N/A

Audit Summary

Version	Delivery Date	Changelog
v1.0	25. July 2023	• Layout Project • Automated- /Manual-Security Testing • Summary

Note - The following audit report presents a comprehensive security analysis of the smart contract utilized in the project. This analysis did not include functional testing (or unit testing) of the contract/s logic. We cannot guarantee 100% logical correctness of the contract as it was not functionally tested by us.

File Overview

The project provided us with the files that should be tested in the security assessment. This audit covered the following files listed below with an SHA-1 Hash.

File Name	SHA-1 Hash
contracts/EastWest.sol	7b6f296fa7fc747f36adcc3e9669e1076971cfbd

Please note: Files with a different hash value than in this table have been modified after the security check, either intentionally or unintentionally. A different hash value may (but need not) be an indication of a changed state or potential vulnerability that was not the subject of this scan.

Imported packages

Used code from other Frameworks/Smart Contracts (direct imports).

Dependency / Import Path	Count
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	1

Note for Investors: We only audited contracts mentioned in the scope above. All contracts related to the project apart from that are not a part of the audit, and we cannot comment on its security and are not responsible for it in any way.

External/Public functions

External/public functions are functions that can be called from outside of a contract, i.e., they can be accessed by other contracts or external accounts on the blockchain. These functions are specified using the function declaration's external or public visibility modifier.

State variables

State variables are variables that are stored on the blockchain as part of the contract's state. They are declared at the contract level and can be accessed and modified by any function within the contract. State variables can be defined with a visibility modifier, such as public, private, or internal, which determines the access level of the variable.

Components

📝 Contracts	📚 Libraries	🔍 Interfaces	🎨 Abstract
1	0	0	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐 Public	💰 Payable
18	0

External	Internal	Private	Pure	View
0	10	0	1	8

StateVariables

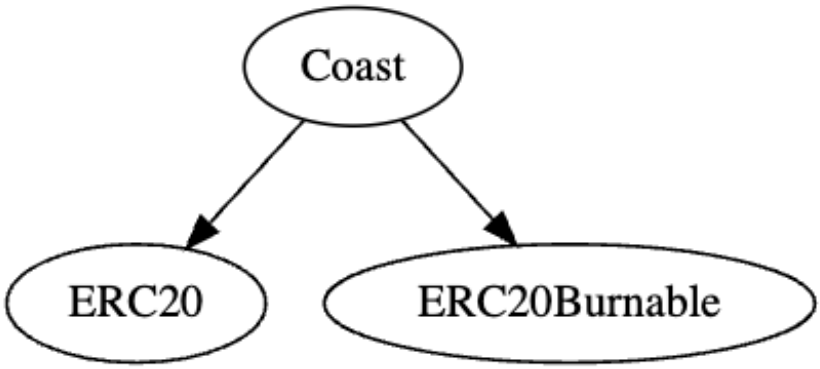
Total	🌐 Public
8	1

Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
`^0.8.0`	----	---	---	----

Inheritance Graph

An inheritance graph is a graphical representation of the inheritance hierarchy among contracts. In object-oriented programming, inheritance is a mechanism that allows one class (or contract, in the case of Solidity) to inherit properties and methods from another class. It shows the relationships between different contracts and how they are related to each other through inheritance.



Audit Information

Vulnerability & Risk Level

Risk represents the probability that a certain source threat will exploit vulnerability and the impact of that event on the organization or system. The risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to check the repository for security-related issues, code quality, and compliance with specifications and best practices. To this end, our team of experienced pen-testers and smart contract developers reviewed the code line by line and documented any issues discovered.

We check every file manually. We use automated tools only so that they help us achieve faster and better results.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
    a. Reviewing the specifications, sources, and instructions provided to SolidProof to ensure we understand the size, scope, and functionality of the smart contract.
    b. Manual review of the code, i.e., reading the source code line by line to identify potential vulnerabilities.
    c. Comparison to the specification, i.e., verifying that the code does what is described in the specifications, sources, and instructions provided to SolidProof.

2. Testing and automated analysis that includes the following:
    a. Test coverage analysis determines whether test cases cover code and how much code is executed when those test cases are executed.
    b. Symbolic execution, which is analysing a program to determine what inputs cause each part of a program to execute.

3. Review best practices, i.e., review smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on best practices, recommendations, and research from industry and academia.

4. Concrete, itemized and actionable recommendations to help you secure your smart contracts.

Overall Security

Medium or higher issues

Upgradeability

Contract is not an upgradeable	✅ **Deployer cannot update the contract with new functionalities**
Description	The contract is not an upgradeable contract. The deployer is not able to change or add any functionalities to the contract after deploying.
Comment	N/A

Ownership

The ownership is not renounced	❌ **The owner is not renounce**
Description	The owner has not renounced the ownership that means that the owner retains control over the contract's operations, including the ability to execute functions that may impact the contract's users or stakeholders. This can lead to several potential issues, including: • Centralizations • The owner has significant control over contract's operations
Comment	N/A

Note - If the contract is not deployed then we would consider the ownership to be not renounced. Moreover, if there are no ownership functionalities then the ownership is automatically considered renounced.

Ownership Privileges

These functions can be dangerous. Please note that abuse can lead to financial loss. We have a guide where you can learn more about these Functions.

Minting tokens

Minting tokens refer to the process of creating new tokens in a cryptocurrency or blockchain network. This process is typically performed by the project's owner or designated authority, who has the ability to add new tokens to the network's total supply.

Contract owner can mint new tokens	❌ **The owner is able to mint new tokens**
Description	Owners who have the ability to mint new tokens can reward themselves or other stakeholders, who can then sell the newly minted tokens on a cryptocurrency exchange to raise funds. However, there is a risk that the owner may abuse this power, leading to a decrease in trust and credibility in the project or platform. If stakeholders perceive that the owner is using their power to mint new tokens unfairly or without transparency, it can result in decreased demand for the token and a reduction in its value.
Example	The Mom address is able to mint unlimited tokens in the COAST contract. Since COAST contract was not the part of the audit, we cannot comment on its security
Comment	N/A

Codebase -

```
162     //mintCoast(_amount): function that mints _amount $CST to the COAST contract and updates available accordingly
163     //  only Mom address can call it
        ftrace | funcSig
164     function mintCoast(uint _amount⬆) public mom_function {
165         available += _amount⬆;
166         _mint(COAST, _amount⬆);
167     }
```

Burning tokens

Burning tokens is the process of permanently destroying a certain number of tokens, reducing the total supply of a cryptocurrency or token. This is usually done to increase the value of the remaining tokens, as the reduced supply can create scarcity and potentially drive up demand.

Contract owner cannot burn tokens	✅ **The owner cannot burn tokens**
Description	The owner is not able burn tokens without any allowances.
Comment	N/A

Blacklist addresses

Blacklisting addresses in smart contracts is the process of adding a certain address to a blacklist, effectively preventing them from accessing or participating in certain functionalities or transactions within the contract. This can be useful in preventing fraudulent or malicious activities, such as hacking attempts or money laundering.

Contract owner cannot blacklist addresses	✅ **The owner cannot blacklist addresses**
Description	The owner is not able blacklist addresses to lock funds.
Comment	N/A

Fees and Tax

In some smart contracts, the owner or creator of the contract can set fees for certain actions or operations within the contract. These fees can be used to cover the cost of running the contract, such as paying for gas fees or compensating the contract's owner for their time and effort in developing and maintaining the contract.

Contract owner cannot set fees more than 25%	✅ **The owner cannot levy unfair taxes**
Description	The owner is not able to set the fees above 25%
Comment	Fee Functionality is ABSENT

Lock User Funds

In a smart contract, locking refers to restricting access to certain tokens or assets for a specified period. When tokens or assets are locked in a smart contract, they cannot be transferred or used until the lock-up period has expired or certain conditions have been met.

Contract owner can lock user tokens in the contract	❌ **The owner is able to lock the user tokens**
Description	Locking the contract means that the owner is able to lock tokens in the contract that they are not able to claim tokens anymore.
Example	An example of locking is by setting the claimable $CST to zero for any address. Moreover, the authority addresses can also turn off the claim manually in which case no one would be able to claim.
Comment	N/A

Codebase -

```
234     function claimableZero(address _claimer⬆) public stacy_function {
235
236         //tmpClaimable: uint value which tracks how much _claimer could claim
237         uint tmpClaimable = claimCoast[currentVersion][_claimer⬆];
238
239         //set claimable $CST for _claimer to 0
240         claimCoast[currentVersion][_claimer⬆] = 0;
241
242         //Check if unclaimed is more than tmpClaimable (it should always be, based on other checks)
243         if (unclaimed > tmpClaimable) {
244             //decrease unclaimed by tmpClaimable
245             unclaimed = unclaimed - tmpClaimable;
246         }
247         else {
248             //setUnclaimed to 0 (this line of code should be impossible to hit)
249             unclaimed = 0;
250         }
251     }
252
253 }
```

Centralization Privileges

Centralization can arise when one or more parties have privileged access or control over the contract's functionality, data, or decision-making. This can occur, for example, if the contract is controlled by a single entity or if certain participants have special permissions or abilities that others do not.

In the project, there are authorities that have access to the following functions:

File	Privileges
Main	❖ mom_function - Enable/Disable Claiming - Increase the current version which will automatically set unclaimed value to zero - Set Stacy Address - Set new MOM address - Mint unlimited tokens ❖ stacy_function - Increase/Decrease the claimable amount for any claimer address - Set the claimable to zero

Recommendations

To avoid potential hacking risks, it is advisable for the client to manage the private key of the privileged account with care. Additionally, we recommend enhancing the security practices of centralized privileges or roles in the protocol through a decentralized mechanism or smart-contract-based accounts, such as multi-signature wallets.

Here are some suggestions of what the client can do:

- Consider using multi-signature wallets: Multi-signature wallets require multiple parties to sign off on a transaction before it can be executed, providing an extra layer of security e.g. Gnosis Safe
- Use of a timelock at least with a latency of e.g. 48-72 hours for awareness of privileged operations
- Introduce a DAO/Governance/Voting module to increase transparency and user involvement
- Consider Renouncing the ownership so that the owner cannot modify any state variables of the contract anymore. Make sure to set up everything before renouncing.

Audit Results

#1 | Missing Timelock

File	Severity	Location	Status
Main	Medium	L116	Open

Description - The contract misses a timelock in the claim function. This means that the claim function can be called recursively.

Remediation - We recommend putting a timelock so that the claim function cannot be called by an external contract recursively and only legitimate users will be able to claim tokens.

#2 | Owner can mint tokens

File	Severity	Location	Status
Main	Medium	L164	Open

Description - The owner of the contract is able to mint unlimited tokens which is not recommended as it may be abused to manipulate the price of the token.

Remediation - Make sure that it is only possible to mint up to a certain max supply or mint all tokens at the time of deployment.

#3 | Owner can lock tokens

File	Severity	Location	Status
Main	Medium	L139, 234	Open

Description - The contract owner can lock tokens and the claim function by disabling it and manually setting the claim amount of users to zero. Moreover, the owner can also increase/decrease the claim amount for any claimer address.

#4 | Missing Events

File	Severity	Location	Status
Main	Low	All	Open

Description - Make sure to emit events for all the critical parameter changes in the contract to ensure the transparency and trackability of all the state variable changes in the contract.

#5 | Missng “isContract” check

File	Severity	Location	Status
Main	Low	L116	Open

Description

- The contract doesn’t have any checks to verify whether the claim function is being called by an EOA or a contract.

Remediation - We recommend putting a check to verify that the callee of the claim function must be an EOA

#6 | Floating Pragma

File	Severity	Location	Status
Main	Informational	L2	Open

Description - The current pragma Solidity directive is “>=0.8.0”. Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using other versions

Legend for the Issue Status

Attribute or Symbol	Meaning
Open	The issue is not fixed by the project team.
Fixed	The issue is fixed by the project team.
Acknowledged(ACK)	The issue has been acknowledged or declared as part of business logic.

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY