



SOLIDProof

Bring trust into your projects

**Blockchain Security | Smart Contract Audits | KYC
Development | Marketing**

MADE IN GERMANY

Carbify Staking

AUDIT

SECURITY ASSESSMENT

02. October, 2024

FOR



SolidProof_io



@solidproof_io

Introduction	4
Disclaimer	4
Project Overview	5
Summary	5
Social Medias	5
Audit Summary	6
File Overview	7
Imported packages	8
Audit Information	9
Vulnerability & Risk Level	9
Auditing Strategy and Techniques Applied	10
Methodology	10
Overall Security	11
Upgradeability	11
Ownership	12
Ownership Privileges	13
Minting tokens	13
Burning Tokens without Allowance	14
Blacklist addresses	15
Fees and Tax	16
Lock User Funds	17
Components	18
Exposed Functions	18
StateVariables	18
Capabilities	19
Inheritance Graph	20
Centralization Privileges	21
Audit Results	23
Critical issues	23
High issues	24



Medium issues	25
Low issues	26
Informational issues	27





Introduction

[SolidProof.io](https://solidproof.io) is a brand of the officially registered company Future Visions Deutschland. We're mainly focused on Blockchain Security, such as Smart Contract Audits and KYC verification for project teams.

Solidproof.io assesses potential security issues in the smart contracts implementations, reviews for potential inconsistencies between the code base and the whitepaper/documentation, and provides suggestions for improvement.

Disclaimer

[SolidProof.io](https://solidproof.io) reports are not, nor should they be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should they be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io does not cover testing or auditing the integration with external contracts 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 analysed, 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 analyse.

Project Overview

Summary

Project Name	Carbify
Website	https://www.carbify.io/
About the project	Carbify is certified by Earthood and, thus, the United Nations and can officially licence/certify projects and companies under The United Nations Framework Convention on Climate Change, which established an international environmental treaty.
Chain	Polygon
Language	Solidity
Codebase Link	Staking Proxy: https://polygonscan.com/address/0x0C666a23304e71294EB2a03b4C27367756e74e01#code Implementation: https://polygonscan.com/address/0x81e391e3e9d9922ae8f0761386dabac2d7b6342f#code Staking Pool Proxy: https://polygonscan.com/address/0x3b8C4c1f01fdc95c7106762D815525aC638b4900#code Implementation: https://polygonscan.com/address/0x0f3e13fd7765c1147d76084002130915f3277575#code
Commit	N/A
Unit Tests	Provided/Not Provided

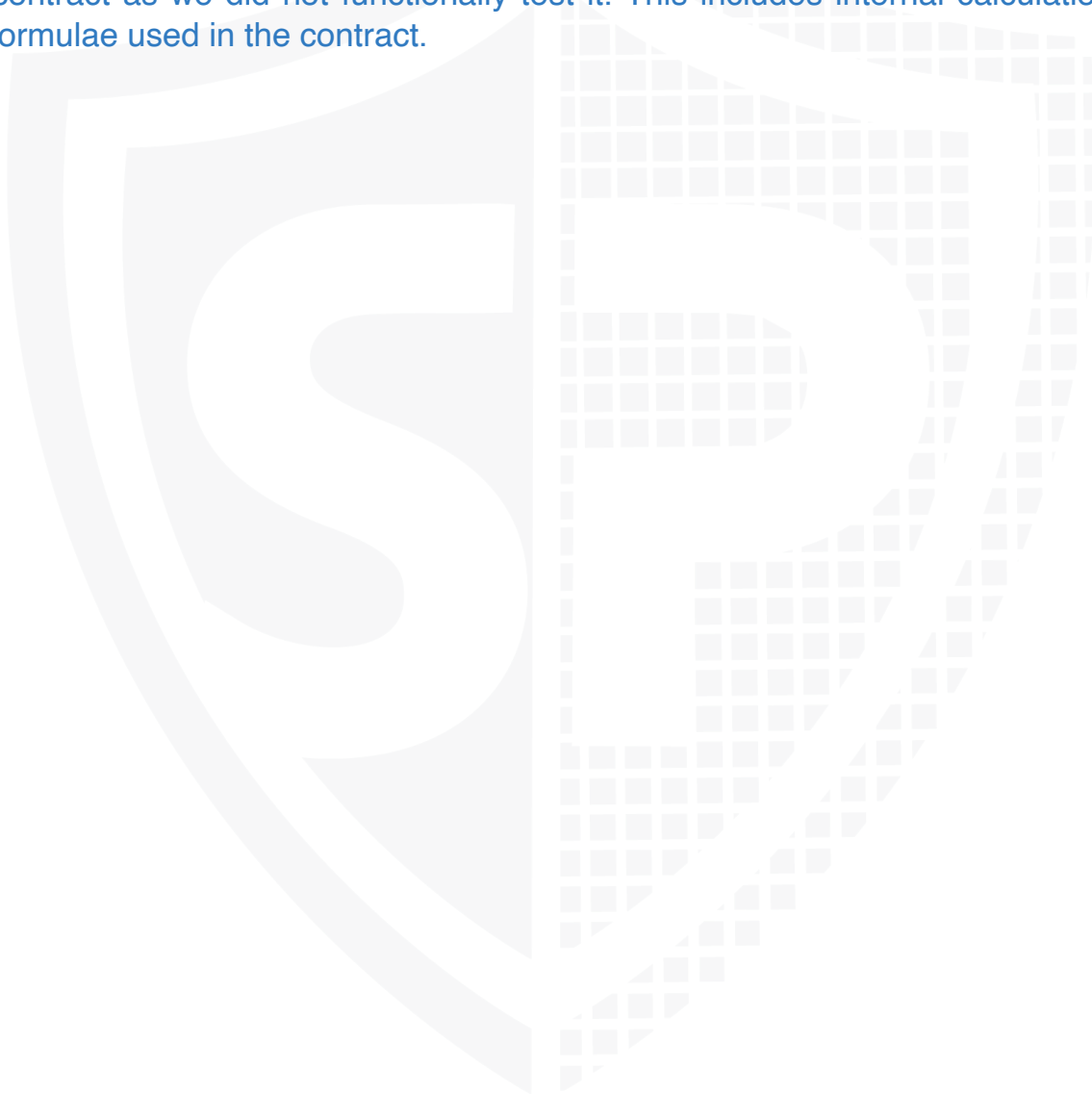
Social Medias

Telegram	https://t.me/carbify
Twitter	https://x.com/carbify_io
Facebook	N/A
Instagram	N/A
Github	N/A
Reddit	N/A
Medium	N/A
Discord	https://discord.com/invite/carbify
Youtube	N/A
TikTok	N/A
LinkedIn	https://linkedin.com/company/carbify/%20

Audit Summary

Version	Delivery Date	Changelog
v1.0	02. October 2024	<ul style="list-style-type: none"> • 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 that includes malicious outside manipulation of the contract's functions. 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 we did not functionally test it. This includes internal calculations in the formulae used in the contract.





File Overview

The Team 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
Staking-2.sol	6b1bd4b42167c47a7d05114a858b9ad600c47c92
StakingPool.sol	09633c91d1ed25d991f7a1080bf15286731b6f2c

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) indicate 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
@cryptoalgebra/core/contracts/interfaces/pool/ IAlgebraPoolState.sol	1
@openzeppelin/contracts-upgradeable/access/ AccessControlUpgradeable.sol	2
@openzeppelin/contracts-upgradeable/proxy/utils/ Initializable.sol	2
@openzeppelin/contracts-upgradeable/proxy/utils/ UUPSUpgradeable.sol	2
@openzeppelin/contracts/token/ERC1155/ IERC1155Receiver.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.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

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 as possible.
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. Review the specifications, sources, and instructions provided to SolidProof to ensure we understand the smart contract's size, scope, and functionality.
 - 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 is analysing a program to determine what inputs cause each part of a program to execute.
3. Review best practices, i.e., 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

Upgradeability

Contract is an upgradeable

✗ Deployer can update the contract with new functionalities

Description

The deployer can replace the old contract with a new one with new features. Be aware of this, because the owner can add new features that may have a negative impact on your investments.

Example

We assume that you have funds in the contract and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.

Comment

N/A

File/Line(s): L58-76

Codebase: Staking-2.sol

```
fttrace | funcSig
function initialize(
    address _cbyTokenAddress↑,
    address _landplotsAddress↑,
    address _landplotsV3Address↑,
    address _landplotsV5Address↑,
    address _algebraPoolAddress↑,
    address _stakingPoolAddress↑
) public initializer {
    __UUPSUpgradeable_init();
    __AccessControl_init();
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(CARBIFY_ADMIN_ROLE, msg.sender);
    cbyToken = IERC20(_cbyTokenAddress↑);
    landplots = ILandplot(_landplotsAddress↑);
    landplotsV3 = ILandplot(_landplotsV3Address↑);
    landplotsV5 = ILandplot(_landplotsV5Address↑);
    algebraPool = IAlgebraPoolState(_algebraPoolAddress↑);
    stakingPool = StakingPool(_stakingPoolAddress↑);
}
```

File/Line(s): L32-39

Codebase: StakingPool.sol

```
fttrace | funcSig
function initialize(address _aco2TokenAddress↑) public initializer {
    __AccessControl_init();
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(STAKING_CONTRACT_ROLE, msg.sender);
    __UUPSUpgradeable_init();
    aco2Token = aC02Token(_aco2TokenAddress↑);
    oldestNonEmptyPackage = 0; // Initialize pointer to the first package
}
```

Ownership

The ownership is not renounced

✗ The owner is not renounce

Description	<p>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:</p> <ul style="list-style-type: none"> • Centralizations • The owner has significant control over contract's operations
Example	<p>We assume that you have funds in the contract and it has been audited by any security audit firm. Now the audit has passed. After that, the deployer can upgrade the contract to allow him to transfer the funds you purchased without any approval from you. This has the consequence that your funds can be taken by the creator.</p>
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 refers 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 can add new tokens to the network's total supply.

Contract owner cannot mint new tokens ✓ The owner cannot mint new tokens	
Description	The owner is not able to mint new tokens once the contract is deployed.
Comment	N/A



Burning Tokens without Allowance

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 contract's cost, 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

N/A

Lock User Funds

In a smart contract, locking refers to the process of restricting access to certain tokens or assets for a specified period of time. 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 the user funds

✗ The owner is able to lock the contract

Description	Locking the contract means that the owner is able to lock any funds of addresses that they are not able to transfer bought tokens anymore.
Example	An example of locking is by pausing the contract or blacklisting any addresses. That causes that the blacklisted address is not able to transfer (buy/sell) anymore.
Comment	The admin can update any arbitrary value in the max token ids per transaction including zero which is not recommended as this can lock the staking and un-staking of the token if the value is set to zero in the contract. There must be a check where this value cannot be set to zero.

File, Line/s: L47-49
Codebase: StakingPool.sol

```
ftrace | funcSig
function setMaxaC02TokenIds(uint256 _maxTokenIds↑) public onlyRole(DEFAULT_ADMIN_ROLE) {
    MAX_TOKEN_IDS_PER_TX = _maxTokenIds↑;
}
```

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
2	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
51	0


External	Internal	Private	Pure	View
27	39	1	2	19

StateVariables

Total	 Public
23	22



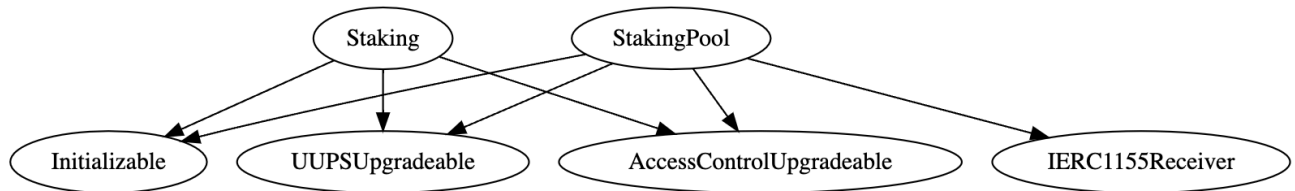
Capabilities

 Solidity Versions observed	Transfers ETH	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
^0.8.20 ^0.8.2				

 Transfers ETH	 Low-Level Calls	 Delegate Call	 Uses Hash Functions	 ECRecover	 New/Create/Create2
yes			yes		

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.



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 a single entity controls the contract or if certain participants have special permissions or abilities that others do not.

In the project, some authorities have access to the following functions:

File	Privileges
Staking-2.sol	<ul style="list-style-type: none"> • Default Admin role <ul style="list-style-type: none"> • Can update carbify admin role. • Can revoke carbify admin role. • Carbify Admin role <ul style="list-style-type: none"> • Can update staking pool contract address. • Can update Algebra pool contract address. • Can update unlock fee receiver address. • Can unlock NFT for the user. • Can unstake NFT for multiple users. • Can claim NFT for the user. • Can update the merle root in the contract. • NFTREE Batch role <ul style="list-style-type: none"> • Can lock the single NFT Tree. • Can lock multiple NFT at once. • Can unlock single or multiple NFT from the contract. • Can stake single NFT or multiple NFT in the contract. • Can unstake single or multiple NFT in the the contract. • Can claim tokens for the users. • LANDPLOT role <ul style="list-style-type: none"> • Can stake single NFT in the contract.
StakingPool.sol	<ul style="list-style-type: none"> • Default Admin role <ul style="list-style-type: none"> • Can update the aco2 token contract address. • Can update the Max token ids per transaction value to any arbitrary value. • Can grant or revoke the staking contract role to any arbitrary address. • Can add the packages in the contract. • Can adjust the package pointers in the contract. • Staking contract role <ul style="list-style-type: none"> • Can claim the CO2 tokens from the contract.



Recommendations

To avoid potential hacking risks, the client should 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 can no longer modify any state variables of the contract. Make sure to set up everything before renouncing.



Audit Results

Critical issues

No critical issues





High issues

No high issues



Medium issues

#1 | The owner can lock staking.

File	Severity	Location	Status
StakingPool.sol	Medium	L47-49	Open

Description - The admin can update any arbitrary value in the max token ids per transaction including zero which is not recommended as this can lock the staking and un-staking of the token if the value is set to zero in the contract. There must be a check where this value cannot be set to zero.

Remediation - Add a 'require' check that the max token per transaction should be more than zero.

#2 | Missing 'isContract' check.

File	Severity	Location	Status
Staking-2.sool	Medium	L90-92, L94-96	Open
StakingPool.sol	Medium	L43-45, L51-53	Open

Description - The contract contains the functionality in which the admin of the contract can update any arbitrary address as the contract address which is not recommended as this can cause the failure of functionality if the address is set to any arbitrary address in the contract. There must be a check where the address can only be set to the contract address to avoid this scenario in the contract.

Remediation - Add a 'require' check that the address can be set to only the contract address.

#3 | Onchain price calculation.

File	Severity	Location	Status
StakingPool.sol	Medium	L102-114	Open

Description - The contract contains the functionality in which the price calculation in the contract is on-chain calculation which is not recommended as on-chain computations in Solidity are expensive, especially for complex logic. Also, on-chain price calculations are vulnerable to manipulation or attacks if not implemented carefully. For example, using unreliable or single-source data on-chain can expose the contract to oracle attacks.

Remediation - off-chain computation paired with trusted oracles for price feeds (e.g., Chainlink) is a more practical and secure approach than performing complex calculations directly on-chain.

Low issues

#1 | Missing Zero Address Validation

File	Severity	Location	Status
Staking-2.sol	Low	L98-100	Open

Description - Make sure to validate that the address passed in the function parameters is “non-zero” or dead address.



Informational issues

#1 | NatSpec documentation missing

File	Severity	Location	Status
All	Informational		Open

Description - If you started to comment on your code, comment on all other functions, variables etc.

#2 | Floating Pragma solidity version

File	Severity	Location	Status
All	Informational	L2	Open

Description - The contracts should be deployed with the same compiler version and flag 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