SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: CarbonPath
Date: March 16, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for CarbonPath
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token; ERC721 token; Token Sale
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.carbonpath.io/
Changelog	22.02.2023 - Initial Review 16.03.2023 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by CarbonPath (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is review and security analysis of smart contracts in the repository:

Initial review scope

Repository	https://github.com/orgs/carbonpathio/repositories
Commit	13a77bc72f4ec5a318031c2aa543a216039ae0d2
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/CarbonPathAdmin.sol SHA3: e27e4e6b321abf67bed5a885cb09c2857e42b113ca9cfd8a03ae308c77459087 File: ./contracts/CarbonPathNFT.sol SHA3: 3bcf184946bfa364980a2eefdc3348d519a33125ab854f8821e06fb257fd35d0 File: ./contracts/CarbonPathToken.sol SHA3: 3511b059224c6569571cd384d491d0b3cb7f0d761d2fd68ff33999da563827ea

Second review scope

Repository	https://github.com/orgs/carbonpathio/repositories
Commit	4e406bb3386b583f630e41b9e26bdbf09e70ea4d
Functional Requirements	https://docs.carbonpath.io/smart_contract.html
Technical Requirements	https://docs.carbonpath.io/smart_contract_architecture.html
Contracts	File: ./contracts/CarbonPathAdmin.sol SHA3: 885face4ac6f0d0e54f15832aac75881ae940857cbbe7d2a54b1847ffff6fb63 File: ./contracts/CarbonPathNFT.sol

	SHA3: 45d387e83d8d768a5797286a243ee31fcc2e9ed4d7df418bb38cfdffa3e3cde5 File: ./contracts/CarbonPathToken.sol SHA3: 8bd9111a27fce8090b86c36110b1c0a60be6fa9fc0723f58438f420bf2ee7a9c

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **7** out of **10**.

- There are some gaps in the functional requirements (there is no explanation of the buffer pool and advanced mint amounts of the CPCO2 token).
- Technical requirements are partially provided.
- NatSpec comments are detailed.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- Code is well-written and designed.

Test coverage

Code coverage of the project is **100%** (branch coverage).

Security score

As a result of the audit, the code contains no issue. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.7**.

1
2
3
4
5
6
7
8
9
10
The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
22 February 2023	10	6	2	2
16 March 2023	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed

Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

CarbonPath is a system that helps to close old oil and natural gas wells. Each unique NFT token of the system represents a real-life well and each fungible token equals a tonne of sequestered carbon emissions. The protocol is designed to allow users to take part in the retirement of abandoned and orphaned wells that pollute the environment.

The files in the audit scope:

- **CarbonPathAdmin** - a contract that handles different interactions with **CarbonPathToken (CPCO2)** and **CarbonPathNFT** and serves as an admin for both contracts. Users can buy CPC02 tokens and retire them for a certain well.
- **CarbonPathNFT** - ERC721 token that bounds to a well. Contains the geolocation of a well and additional metadata.
- **CarbonPathToken** - ERC20 mintable and burnable token that represents locked carbon emission.

Privileged roles

- DefaultAdmin:
    - CarbonPathAdmin: can grant and revoke minter role, set receiver of CP fee, non-profit address, buffer pool address, seller address.
    - CarbonPathToken: can grant and revoke minter role
- Owner:
    - CarbonPathNFT: can set admin address
- Admin:
    - CarbonPathNFT: can set metadata and URI for a specific token, mint new tokens, and increase the amount of retired CPCO2 tokens
- Minter:
    - CarbonPathAdmin: can mint new tokens, update token URI and metadata
    - CarbonPathToken: can mint, burn tokens

Risks

- The system is not adequately documented. It is difficult to determine compliance with requirements due to gaps in the requirements.
- The system describes the sellerAddress (single address) used in the CarbonPathAdmin contract as the CarbonPath Wallets; the security of the CarbonPath Wallets functionality is not part of this audit.

Recommendations

- The system relies on the security of the privileged roles’ private keys, which can impact the execution flow and security of the funds. We recommend those accounts to be at least ⅗ multi-sig.

Findings

■■■■ Critical

C01. Requirements Violation

The *cpFeeAddress*, *bufferPoolAddress,* and *nonProfitAddress* variables are not explicitly set in the constructor and have a default value of *address(0)*. In such a case, during the call of the *mint()* function, the fees and buffer tokens will be transferred to address(0).

Path:
./contracts/CarbonPathAdmin.sol

Recommendation: Set the *cpFeeAddress*, *bufferPoolAddress,* *nonProfitAddress* values in the constructor.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

C02. Requirements Violation

The name of the internal function *_isAdminOrOwner()* contradicts its functionality, which also allows approved addresses to perform the same operations.

It also contradicts the require error message:

"CarbonPathNFT: must be an admin or an owner"

Path:
./contracts/CarbonPathAdmin.sol : _isAdminOrOwner()

Recommendation: Consider updating the code to comply with the documentation.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

■■■ High

H01. Highly Permissive Role Access

The CarbonPathNFT has URI and Metadata that can be changed by the owner of the token or the admin of the system.

This can lead to unwanted changes to the state of the NFT.

Path:
./contracts/CarbonPathNFT.sol : setMetadata(), setTokenURI()

Recommendation: Reconsider whether the user can change their token data, or make this functionality available only to the admin. The second option requires a publicly available document with a description of such functionality for the users.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Mitigated (The data URI and Metadata can be changed only by the Admin role. And the option to change the URI or the Metadata is required by the system. This functionality is documented in the provided documentation as:

"NOTE: CarbonPath may update the metadata and well files even after the well is minted".)

H02. Undocumented Behavior

The system uses different CarbonPathToken distributions during the function calls, which are not documented in the requirements.

Path:
./contracts/CarbonPathAdmin.sol : mint()

Recommendation: Add documentation about funds flow and the distribution of tokens.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Mitigated (Documentation was updated with the description of the funds flow:

"During minting, CPCO2 Tokens are airdropped to the following recipients:

- cpFeeAddress will receive advancedAmount * cpFeePercentage of CPCO2 Tokens.
- operatorAddress will receive advancedAmount * (100% - cpFeePercentage) of CPCO2 Tokens.
- bufferPoolAddress will receive bufferPoolAmount * 95% of CPCO2 Tokens.
- nonProfitAddress will receive bufferPoolAmount * 5% of CPCO2 Tokens.

".)

Medium

M01. Unscalable Functionality: Check Repetition

The same check *require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()))* used in several functions overwhelms the code and makes further development difficult.

Path:
./contracts/CarbonPathAdmin.sol

Recommendation: Move the check to special modifiers.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

M02. Unscalable Functionality: Check Repetition

The same check *require(_address != address(0))* used in several functions overwhelms the code and makes further development difficult.

Path:
./contracts/CarbonPathAdmin.sol

Recommendation: Move the check to special modifiers.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

M03. Unscalable Functionality: Check Repetition

The check *require(amount > 0)* used in several functions overwhelms code and makes further development difficult.

Path:
./contracts/CarbonPathAdmin.sol

Recommendation: Move the check to special modifiers.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

M04. Inefficient Gas Model: SafeMath in Solidity ^0.8.0

Starting with Solidity ^0.8.0, SafeMath functions are built-in. This makes the library redundant.

Paths:
./contracts/CarbonPathAdmin.sol
./contracts/CarbonPathNFT.sol

Recommendation: Remove the SafeMath library.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

M05. Contradiction: Redundant Ownable

The Ownable in CarbonPathAdmin is inherited but not used and contradicts the usage of the AccessControl contract.

Path:
./contracts/CarbonPathAdmin.sol

Recommendation: Remove inheritance from the Ownable contract.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

M06. Inconsistent Data: Missing Event for Critical Value Update

The `_adminAddress` is updated in the constructor() and in setAdminAddress() but the corresponding event is not emitted.

Path:
./contracts/CarbonPathNFT.sol

Recommendation: Emit the event whenever the corresponding action happens.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

Low

L01. Gas Optimization: Long Error Strings

The error strings in *require* statements are longer than 32 bytes.

This increases contract size and Gas usage.

Paths:
./contracts/CarbonPathAdmin.sol
./contracts/CarbonPathNFT.sol
./contracts/CarbonPathToken.sol

Recommendation: Use shorter error strings.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

L02. Style Guide Violation

The project should follow the official code style guidelines.
Inside each contract, library, or interface, use the following order:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public

- internal
- private

Within a grouping, place the view and pure functions at the end.

Paths:
./contracts/CarbonPathAdmin.sol
./contracts/CarbonPathNFT.sol
./contracts/CarbonPathToken.sol

Recommendation: The official Solidity style guidelines should be followed.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

L03. Functions that Can Be Declared External

In order to save Gas, *public* functions that are never called in the contract should be declared as *external*.

Paths:
./contracts/CarbonPathAdmin.sol : grantMinter(), revokeMinter(), setCpFeeAddress(), setSellerAddress(), mint(), updateTokenURI(), updateMetadata(), retire(), sell(), withdraw(), buy()
./contracts/CarbonPathNFT.sol : getAdminAddress(), getAdvancedEAVs(), getBufferPoolEAVs(), getRetiredEAVs(), getGeoJson(), getMetadata(), setAdminAddress(), setMetadata(), setTokenURI(), mint(), updateRetiredEAVs()
./contracts/CarbonPathToken.sol : grantMinter(), revokeMinter(), mint(), burn(), burnFrom()

Recommendation: Use the *external* attribute for functions that are never called from the contract.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

L04. Unfinished NatSpec

It is recommended that the code should be kept clean and properly documented with NatSpec. There are multiple functions, structs, and public storage variables that do not have proper NatSpec documentation.

Paths:
./contracts/CarbonPathAdmin.sol
./contracts/CarbonPathNFT.sol
./contracts/CarbonPathToken.sol

Recommendation: NatSpec documentation best practices should be followed. For reference:

https://docs.soliditylang.org/en/v0.8.18/natspec-format.html#documentation-example
https://dev.to/perelynsama/natspec-the-right-way-to-comment-ethereum-smart-contracts-1b0c

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

L05. Unclear Use of the Virtual Specifier

There are functions in the contracts that are declared with the virtual specifier. These functions are not expected to be overridden, so the use of the virtual specifier is redundant.

Paths:
./contracts/CarbonPathAdmin.sol : mint(), updateTokenURI(), updateMetadata(), retire(), sell(), withdraw(), buy()
./contracts/CarbonPathNFT.sol : mint(), updateRetiredEAVs()
./contracts/CarbonPathToken.sol : burn(), burnFrom()

Recommendation: Remove the `virtual` modifier from functions of top-level contracts.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

L06. Missing Zero Address Validation

Address parameters are used without checking against the possibility of being 0x0.

This can lead to unwanted external calls to 0x0.

Paths:
./contracts/CarbonPathAdmin.sol : grantMinter()
./contracts/CarbonPathToken.sol : grantMinter()

Recommendation: Implement zero address validations.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

L07. Redundant nonReentrant Modifier

In all functions (with one small exception), the CEI pattern is followed, thus reducing the risk of reentrancy.

The use of the nonReentrant modifier in all functions looks redundant.

This makes the import of ReentrancyGuard and inherit from it as unnecessary.

Paths:
./contracts/CarbonPathAdmin.sol : setSellerAddress(), mint(), retire(), sell(), withdraw(), buy()
./contracts/CarbonPathNFT.sol : mint()

Recommendation: Consider removing redundant code.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

L08. Checks-Effects-Interactions Pattern Violation

During the *setSellerAddress()* function execution, the sellerAddress state variable is updated after the external calls.

There is no direct risk involved, but it is best practice to always follow the CEI pattern.

Path:
./contracts/CarbonPathAdmin.sol : setSellerAddress()

Recommendation: Common best practices should be followed, update the state variable before the external contract call.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

L09. Typo in Comments

There are multiple spelling errors in the comments:

receipients -> recipients
Tranfer -> Transfer
tranferred -> transferred
CarbonpathAdmin -> CarbonPathAdmin

Path:
./contracts/CarbonPathAdmin.sol

Recommendation: Spellings should be fixed.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

L10. Redundant Import

In the CarbonPathToken contract, the Pausable contract is imported but never used.

The use of unnecessary imports will increase the Gas consumption of the code. Thus, they should be removed from the code.

Redundant imports decrease code readability.

Path:
./contracts/CarbonPathToken.sol

Recommendation: Consider removing redundant code.

Found in: 13a77bc72f4ec5a318031c2aa543a216039ae0d2

Status: Fixed (Revised commit: 4e406bb)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with 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).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

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 recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.