



Carbon Browser

Token

SMART CONTRACT AUDIT

05.01.2023

Made in Germany by Chainsulting.de



Table of contents

1. Disclaimer.....	3
2. About the Project and Company	4
2.1 Project Overview.....	5
3. Vulnerability & Risk Level	6
4. Auditing Strategy and Techniques Applied.....	7
4.1 Methodology	7
5. Metrics	8
5.1 Tested Contract Files	8
5.2 Used Code from other Frameworks/Smart Contracts	9
5.3 CallGraph	10
5.4 Inheritance Graph	11
5.5 Source Lines & Risk	12
5.6 Capabilities	13
5.7 Source Unites in Scope	14
6. Scope of Work.....	15
6.1 Findings Overview	16
6.2 Manual and Automated Vulnerability Test.....	17
6.3 SWC Attacks	18
6.4. Verify Claims	22
7. Executive Summary.....	23
8. Deployed Codebase.....	23
9. About the Auditor	24

1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Carbon X Labs Ltd. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (03.01.2023)	Layout
0.4 (04.01.2023)	Automated Security Testing Manual Security Testing
0.5 (04.01.2023)	Verify Claims and Test Deployment
0.6 (05.01.2023)	Testing SWC Checks
0.9 (05.01.2023)	Summary and Recommendation
1.0 (05.01.2023)	Final document

2. About the Project and Company

Company address:

CONFIDENTIAL



Website: <https://carbon.website>

Twitter: <https://twitter.com/trycarbonio>

Telegram: <https://t.me/trycarbonio>

Blog: <https://carbon.website/news/>

GitHub: <https://github.com/Carbon-Browser>

2.1 Project Overview

Carbon is a free and open-source web browser developed by Carbon X Labs based on a custom fork of the Chromium SDK and its powerful Blink engine.

Carbon is the go-to browser for Android users who value fast performance, security, and privacy. It automatically blocks online ads and website trackers to provide a private and secure browsing experience. Carbon Browser aims to solve the issues of decentralization, lack of privacy, and slow loading speeds. They do it by focusing on web3 needs, giving users those critical features in an easy-to-use platform.

With Carbon, you can completely control your online presence with a multi-chain wallet, giving you the power to manage all of your online activity securely. Use web 3.0's bridge function to connect web 2.0 and web 3.0 ecosystems, allowing for a smarter and more streamlined experience.

3. 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. 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

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
 - ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
 - iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
 - i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
 - ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (MD5)
./contracts/CarbonToken.sol	42147c109aa105c49a733cc8fcbecfc0
./contracts/CarbonTokenFee.sol	0e30ce3e66403e870f515ae871c4842d

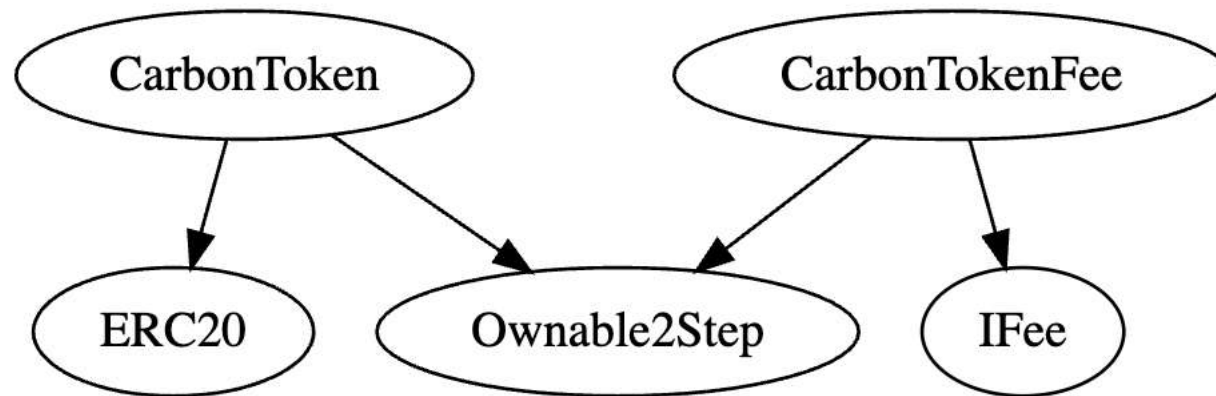
5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable2Step.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.0/contracts/access/Ownable2Step.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.8.0/contracts/token/ERC20/ERC20.sol

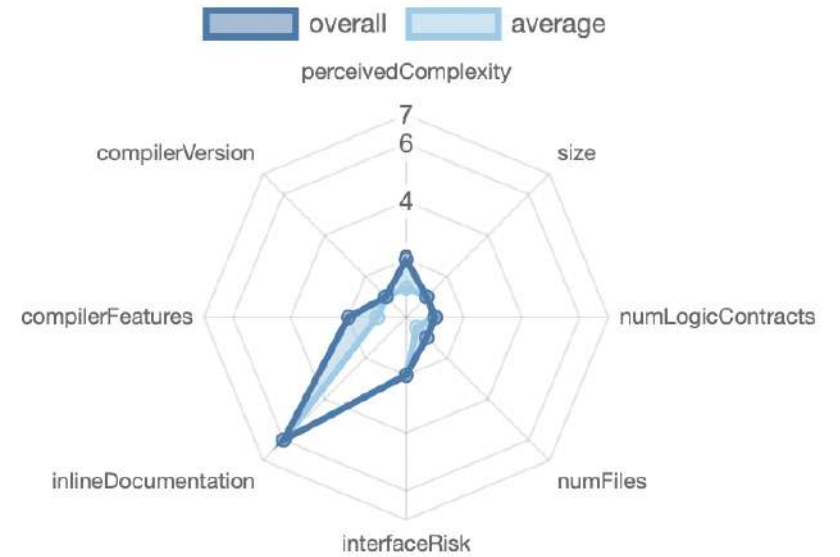
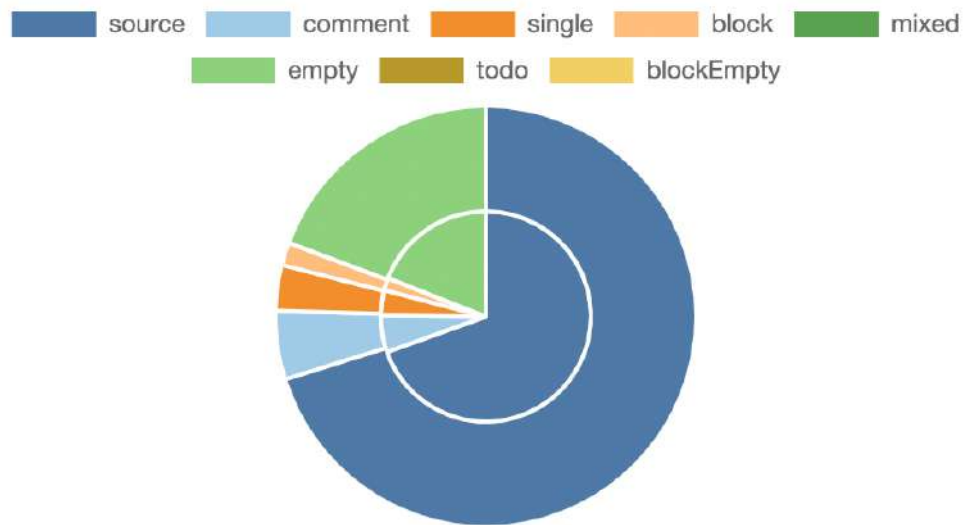
5.3 CallGraph













5.4 Inheritance Graph



5.5 Source Lines & Risk





5.6 Capabilities


Solidity Versions observed		 Experimental Features		 Can Receive Funds		 Uses Assembly		 Has Destroyable Contracts			
0.8.17											
 Transfers ETH		 Low-Level Calls		 DelegateCall		 Uses Hash Functions		 ECRrecover		 New/Create/Create2	
yes											

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				
11	0				
External	Internal	Private	Pure	View	
2	10	0	0	3	

StateVariables

Total	 Public
4	4

5.7 Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/CarbonToken.sol	1		57	53	40	3	49	
	contracts/CarbonTokenFee.sol	1	1	51	46	33	3	38	
	Totals	2	1	108	99	73	6	87	

Legend:

- **Lines:** total lines of the source unit
- **nLines:** normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC:** normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines:** lines containing single or block comments
- **Complexity Score:** a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

6. Scope of Work

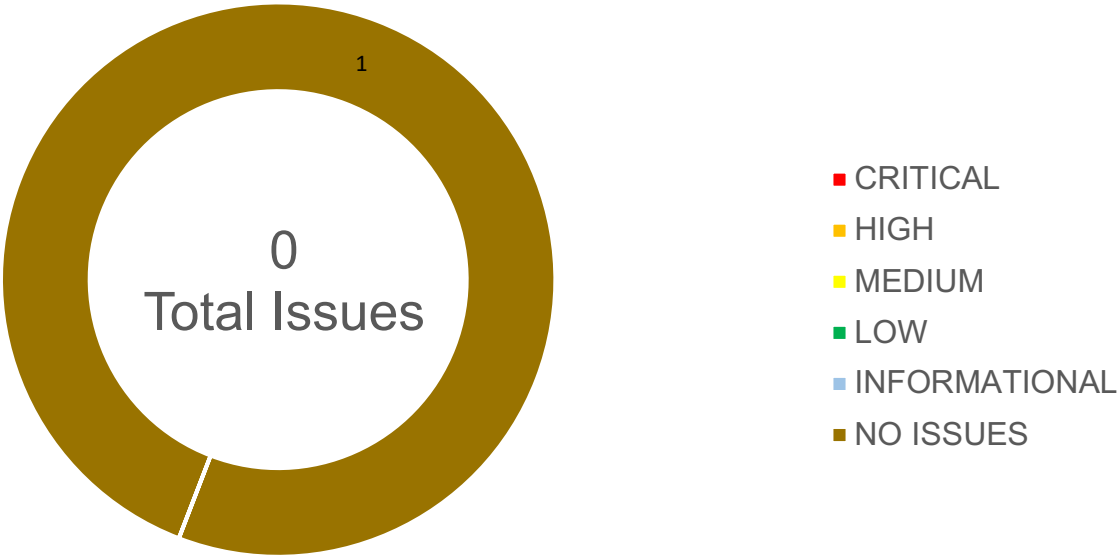
The Carbon Browser Team provided us with the files that needs to be tested. The scope of the audit is the Carbon Token contract.

The team put forward the following assumptions regarding the security, usage of the contracts:

- The contract is using the ERC-20 token standard
- Owner cannot mint new tokens
- Owner cannot burn or lock user funds
- Owner cannot pause the contract
- The smart contract is coded according to the newest standards and in a secure way

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

6.1 Findings Overview



No	Title	Severity	Status
----	-------	----------	--------

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting's experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, Chainsulting's experts found **no Medium issues** in the code of the smart contract

LOW ISSUES

During the audit, Chainsulting's experts found **no Low issues** in the code of the smart contract

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **no Informational issues** in the code of the smart contract

6.3 SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✓
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✓
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✓
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✓
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✓
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✓
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✓
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✓


ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✓
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✓
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✓
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✓
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✓
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✓

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✓
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✓
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✓
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✓
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✓
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✓
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✓
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✓


ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✓
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✓
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✓
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓

6.4. Verify Claims


6.4.1 The contract is using the ERC-20 token standard

Status: tested and verified 


6.4.2 Owner cannot mint new tokens

Status: tested and verified 


6.4.3 Owner cannot burn or lock user funds

Status: tested and verified 

6.4.4 Owner cannot pause the contract

Status: tested and verified 

6.4.5 The smart contract is coded according to the newest standards and in a secure way.

Status: tested and verified 

7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase.

The main goal of the audit was to verify the claims regarding the security and functions of the smart contract. During the audit, no critical, no high, one medium, no low and no informational issues have been found, after the manual and automated security testing.

8. Deployed Codebase

Carbon Token: <https://bscscan.com/address/0x04756126f044634c9a0f0e985e60c88a51acc206#code>

Carbon Fees: <https://bscscan.com/address/0x111541df26BeD3BcBEa90463bBB46F89fd95Ec1E#code>



9. About the Auditor

Chainsulting is a professional software development firm, founded in 2017 and based in Germany. They show ways, opportunities, risks and offer comprehensive Web3 solutions. Their services include web3 development, security and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Solana, Tezos, Ethereum, Binance Smart Chain, and Polygon to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secure the smart contracts of many top DeFi projects.

Chainsulting currently secures [\\$100 billion](#) in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the web3 sector to deliver top-notch smart contract audit solutions, tailored to the clients' evolving business needs.

Check our website for further information: <https://chainsulting.de>

How We Work



1 -----

PREPARATION

Supply our team with audit ready code and additional materials



2 -----

COMMUNICATION

We setup a real-time communication tool of your choice or communicate via e-mails.



3 -----

AUDIT

We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.



4 -----

FIXES

Your development team applies fixes while consulting with our auditors on their safety.



5 -----

REPORT

We check the applied fixes and deliver a full report on all steps done.