Blockchain Security & Audits

Seculite

Blockchain Security Company

CareBig Sale Audit

Security Assesment

24. January 2023

For

Seculite.net

SeculiteAudits

Disclaimer 2
Description 4
Project Engagement 4
Logo 4
Contract Link 4
Methodology 6
Used Code from other Frameworks/Smart Contracts (direct imports) 7
Tested Contract Files 8
Source Lines 9
Risk Level 9
Capabilities 10
Inheritance Graph 11
CallGraph 12
Scope of Work/Verify Claims 13
Modifiers and Public Functions 18
Source Units in Scope 20
Critical issues 21
High issues 21
Medium issues 21
Low issues 21
Informational issues 21
Audit Comments 22
SWC Attacks 23

Disclaimer

Seculite Solutions (Seculite) 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. Seculite do not cover testing or auditing the integration with external contracts or services (such as Unicrypt, Uniswap, PancakeSwap etc’...)

Seculite 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. Seculite 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.

Seculite 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. Seculite’s position is that each company and individual are responsible for their own due diligence and continuous security. Seculite in no way claims any guarantee of the security or functionality of the technology we agree to analyse.

Version	Date	Description
1.0	24 January 2023	• Layout project • Automated- /Manual-Security Testing • Summary

Network

Binance Smart Chain (BNB)

Website

https://carebigtoken.io

Description

TBA

Project Engagement

During the Date of 16 January 2023, **CareBig Team** engaged Seculite to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Seculite with access to their code repository and whitepaper.

Logo

CAREBIG
TOKEN

Contract Link

v1.0

- Contract: https://github.com/MasterAtWork/carebig_smartcontracts/blob/CB-66/contracts/CareBigSale.sol
- Commit: cab976d8234de863b0d2b18ffb8e6e65e9cf9698TBA

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	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 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.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
    a. Review of the specifications, sources, and instructions provided to Seculite to make sure we understand the size, scope, and functionality of the smart Contract.
    b. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    c. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Seculite describe.

2. Testing and automated analysis that includes the following:
    a. 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.
    b. 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, itemised, actionable recommendations to help you take steps to secure your smart contracts.

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

Imported Packages:

Dependency / Import Path	Count
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/security/ReentrancyGuard.sol	1
@openzeppelin/contracts/security/Pausable.sol	1
@openzeppelin/contracts/access/AccessControl.sol	1

Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

File Name	SHA-1 Hash
contracts/CareBigSale.sol	7494d891ada8568b34554cde00efa38b1 3702758

Metrics

Source Lines

V 1.0

Risk Level

V 1.0

Capabilities

Contracts	Libraries	Interfaces	Abstract
1	0	1	0

Exposed Functions

This section lists functions that are explicitly declared public or private.

Public	Payable
2	0

External	Internal	Private	Pure	View
8	1	0	0	4

State Variables

Total	Public
9	8

Capabilities

Solidity Version observed	Experimental Features	Can Receive Funds	Uses Assembly
0.8.11	-	-	-

Has Destroyable Contracts	Transfer ETH	Low-level calls	DelegateCall
-	-	-	Yes

ECRecover	New/Create/Create2	TryCatch	Unchecked
-	-	-	-

Uses Hash Functions
-

Inheritance Graph

V 1.0

CallGraph

V 1.0

Scope of Work/Verify Claims

The above token Team provided us with the files that needs to be tested(GitHub, Bscscan, Etherscan, les, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Is contract an upgradeable
2. Deployer cannot pause the contract
3. Overall checkup (Smart Contract Security)

Is contract an upgradeable

Name	Answer
Is contract an upgradeable?	No

Write functions of contract

V 1.0

- buy
- pause
- unpause
- setMinTokensToBuy
- setMaxTokensToBuy
- updatedAllowedPaymentToken
- updatePriceOfToken
- updateBeneficiary

Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	✓	✓	✗

Comments

V 1.0

The wallets with the roles “PAUSER” can pause the contracts.

Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

Attribute	Symbol
Verified / Checked	✓
Parity Verified	🚩
Unverified / Not Verified	✗
Not available	-

Modifiers and Public Functions

V 1.0

- buy
  - nonReentrant
- pause
  - onlyRole
- unpause
  - onlyRole
- setMinTokensToBuy
  - onlyRole
- setMaxTokensToBuy
  - onlyRole
- updatedAllowedPaymentToken
  - onlyRole
- updatePriceOfToken
  - onlyRole
- updateBeneficiary
  - onlyRole

Ownership/Authorised Addresses Privileges:

V 1.0

- There are several authorities which are authorized to call certain functions, that means, if the owner is renounced, another address is still authorized to call functions
- “OPERATOR” role can:
    - Set minimum and maximum amount of tokens to buy to any arbitrary value except zero.
    - Set the price of token to any value which will affect the users when they will be buying tokens from the sale contract
- “DEFAULT_ADMIN_ROLE” role can:
    - Update payment tokens, which means this authority can allow/ disallow any tokens by which the CareBig token can be bought during the sale. Moreover, these payment tokens can also be changed Mid-Sale.

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope

V 1.0

File	Logic Contracts	Interfaces	Lines	nLines	nSOLC	Comment Lines	Complex. Score
Contracts/CareBigSale.sol	1	1	197	190	162	8	103
Total	1	1	197	190	162	8	103

Legend

V 1.0

Attribute	Description
Lines	total lines of the source unit
nLines	normalised lines of the source unit (e.g. normalises functions spanning multiple lines)
nSLOC	normalised 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, ...)

Audit Result

Critical issues

No critical issues

High issues

No High risk found

Medium issues

No medium issues

Low issues

Issue	File	Type	Line	Description
#1	CareBigSale.sol	Missing Zero Address Validation (missing-zero-check)	83	Check that the address is not zero

Informational issues

No informational issues

Issue	File	Type	Line	Description
#1	CareBigSale.sol	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc.

Audit Comments

V 1.0

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/ latest/natspec-format.html) for your contracts to provide rich documentation for functions, return variables and more. This helps investors to make clear what that variables, functions etc. do.

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SWC-106	Unprotected SELFDESTRUC T Instruction	CWE-284: Improper Access Control	PASSED
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	PASSED
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

A Blockchain Security Company

Seculite Secured

Seculite.net

SeculiteAudits