AUDITED BY
LOVELY INSPECTOR

SMART CONTRACT

SECURITY AUDIT

COBITX TOKEN

inspector.lovely.finance

LOVELY INSPECTOR

TABLE OF CONTENTS

DISCLAIMER

This is a comprehensive report based on our automated and manual examination of cybersecurity vulnerabilities and framework flaws of the project's smart contract. Reading the full analysis report is essential to build your understanding of the project's security level. It is crucial to take note, though we have done our best to perform this analysis and report, that you should not rely on our research and cannot claim what it states or how we created it. Before making any judgments, you have to conduct your own independent research. We will discuss this in more depth in the following disclaimer - please read it fully.

DISCLAIMER: You agree to the terms of this disclaimer by reading this report or any portion thereof. Please stop reading this report and remove and delete any copies of this report that you download and/or print if you do not agree to these conditions. Scan and verify the report's presence in the GitHub repository by a QR code on the title page. This report is for non-reliability information only and does not represent investment advice. No one shall be entitled to depend on the report or its contents, and Inspector Lovely and its affiliates shall not be held responsible to you or anyone else, nor shall Inspector Lovely provide any guarantee or representation to any person with regard to the accuracy or integrity of the report. Without any terms, warranties, or other conditions other than as set forth in that exclusion Inspector Lovely excludes hereby all representations, warrants, conditions, and other terms (including, without limitation, guarantees implied by the law of satisfactory quality, fitness for purposes and the use of reasonable care and skills). The report is provided as "as is" and does not contain any terms and conditions. Except as legally banned, Inspector Lovely disclaims all responsibility and responsibilities, and no claim against Inspector Lovely is made to any amount or type of loss or damages (without limitation, direct, indirect, special, punitive, consequential, or pure economic loses or losses) that may be caused by you or any other person, or any damages or damages, including without limitations (whether innocent or negligent). Security analysis is based only on the smart contracts. No applications or operations were reviewed for security. No product code has been reviewed.

LOVELY INSPECTOR

AUDIT SCOPE

Name	Code Review and Security Analysis Report for Cobitx Token Smart Contract
Platform	Binance Smart Chain
Language	Solidity
File	SolidityCobitx.sol
Initial code link	0xc7e970e6b2a0b5b775542eb05b5b55edad0768cb
Updated code link	0xcfe75850cb18876be3fdfffcdf05d69844f133a8
Audit Date	February 29th, 2024
Revised Audit Date	March 1st, 2024

LOVELY INSPECTOR

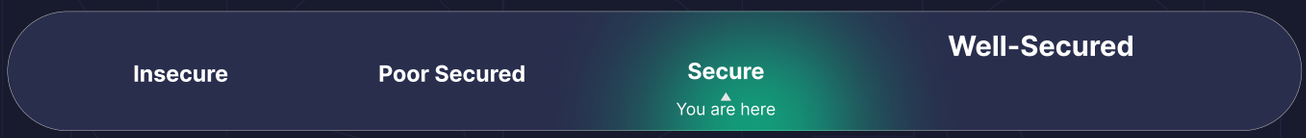
PROPOSED SMART CONTRACT FEATURES

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Cobitx • Symbol: CBTX • Decimals: 18 • Total Supply: 990 million	Validated
Ownership control: • Current owner can transfer ownership of the contract to a new account. • Transfer tokens from supply to the specified address only by the owner.	Validated

LOVELY INSPECTOR

AUDIT SUMMARY

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **“Secured”**. Also, these contracts contain owner control, which does not make them fully decentralized.



We used various tools like Slither, Solhin,t, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 1 low and 2 very low level issues.
We confirm that all severity issues are fixed in the revised smart contract code.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

LOVELY INSPECTOR

KEY TECHNICAL METRICS

MAIN CATEGORY	SUBCATEGORY	RESULT
Contract Programming	Solidity version is not specified	Passed
	Solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

LOVELY INSPECTOR

BUSINESS RISK ANALYSIS

CATEGORY	RESULT
Buy Tax	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	No
Fee Check	Not Detected
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Not Detected
Pause Transfer?	Not Detected
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	No
Is it Proxy?	No
Can Take Ownership?	Yes
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: PASSED

LOVELY INSPECTOR

CODE QUALITY

This audit scope has 1 smart contract. Smart contract contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in Cobitx Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Cobitx Token.

The _____ team has not provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.
Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

DOCUMENTATION

We were given a Cobitx Token smart contract code in the form of a bscscan web link.
As mentioned above, code parts are well commented on. and the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

USE OF DEPENDENCIES

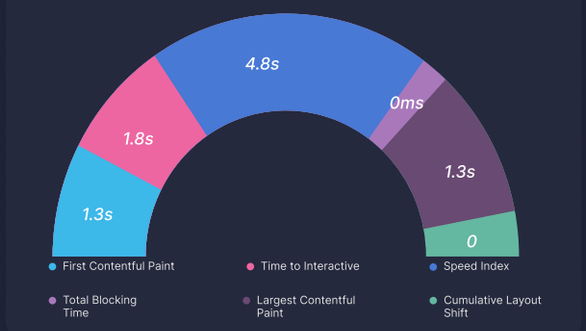
As per our observation, the libraries are used in this smart contract infrastructure that are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

LOVELY INSPECTOR

PROJECT WEBSITE PERFORMANCE AUDIT

Performance Metrics



Browser Timings

Redirect Duration **0ms**	Connection Duration **358ms**	Backend Duration **284ms**
Time to First Byte **642ms**	First Paint **1.3s**	DOM Interactive Time **1.8s**
DOM Content Loaded **1.8s**	Onload Time **3.9s**	Fully Loaded Time **4.2s**

Grade

B	Performance 84%	Structure 81%

Web Vitals

LCP	TBT	CLS
1.3s	0ms	0

Top Issues

IMPECT	AUDIT
Med	Avoid enormous network payloads (LCP)

URL SIZE

LOVELY INSPECTOR

LEVEL OF CRITICALITY

RISK LEVEL	DESCRIPTION
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Med	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

LOVELY INSPECTOR

AS-IS OVERVIEW

Sl	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	onlyOwner	modifier	Passed	No Issue
3	transferTokens	write	access only Owner	No Issue
4	transferOwnership	write	access only Owner	Fixed
5	name	read	Passed	No Issue
6	symbol	read	Passed	No Issue
7	decimals	read	Passed	No Issue
8	total Supply	read	Passed	No Issue
9	balance Of	read	Passed	No Issue
10	transfer	write	Passed	No Issue
11	transfer	read	Passed	No Issue
12	approve	write	Passed	No Issue
13	transferFrom	write	Passed	No Issue
14	increaseAllowance	write	Passed	No Issue
15	decreaseAllowance	write	Passed	No Issue
16	_transfer	internal	Passed	No Issue
17	_mint	internal	Passed	No Issue
18	_burn	internal	Passed	No Issue
19	_approve	internal	Passed	No Issue
20	_spendAllowance	internal	Passed	No Issue
21	_beforeTokenTransfer	internal	Passed	No Issue
22	_afterTokenTransfer	internal	Passed	No Issue
23	_msgSender	internal	Passed	No Issue
24	_msgData	internal	Passed	No Issue

LOVELY INSPECTOR

AUDIT FINDINGS

Critical Severity	No Critical severity vulnerabilities were found.
High Severity	No High severity vulnerabilities were found.
Medium	No Medium severity vulnerabilities were found.
Low	No Low severity vulnerabilities were found.
Very Low / Informational / Best practices:	No Very Low severity vulnerabilities were found.

Status : Fixed

Very Low / Informational / Best practices:

(1) Multiple Pragma:

```
15  pragma solidity ^0.8.0;
16
17 > /** ···
27 > abstract contract Context { ···
35  }
36
37  // File: @openzeppelin/contracts/token/ERC20/IERC20.sol
38
39
40  // OpenZeppelin Contracts (last updated v4.9.0) (token/ERC20/IERC20.sol)
41
42  pragma solidity ^0.8.0;
```

Contract code has multiple pragma solidity versions.

Resolution: We suggest having only one pragma version written at the beginning of the code.

Status: Fixed

```
15  pragma solidity ^0.8.0;
```

LOVELY INSPECTOR

CENTRALIZATION

This smart contract has some functions that can be executed by the Admin (Owner) only. If the admin wallet's private key is compromised, then it would create trouble. Following are Admin functions:

This smart contract has some functions that can be executed by the Admin (Owner) only. If the admin wallet's private key were compromised, then it would create trouble. Following are Admin functions:

Cobitx.sol

- transferTokens: Transfer tokens from supply to the specified address only by the owner.
- transferOwnership: Current owner can transfer ownership of the contract to a new account.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

LOVELY INSPECTOR

CONCLUSION

We were given a contract code in the form of a bscscan web link. And we have used all possible tests based on given objects as files. We observed 1 low and 2 informational issues in the smart contracts. We confirm that all severity issues are fixed in the revised smart contract code. So, it's good to go for the production.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

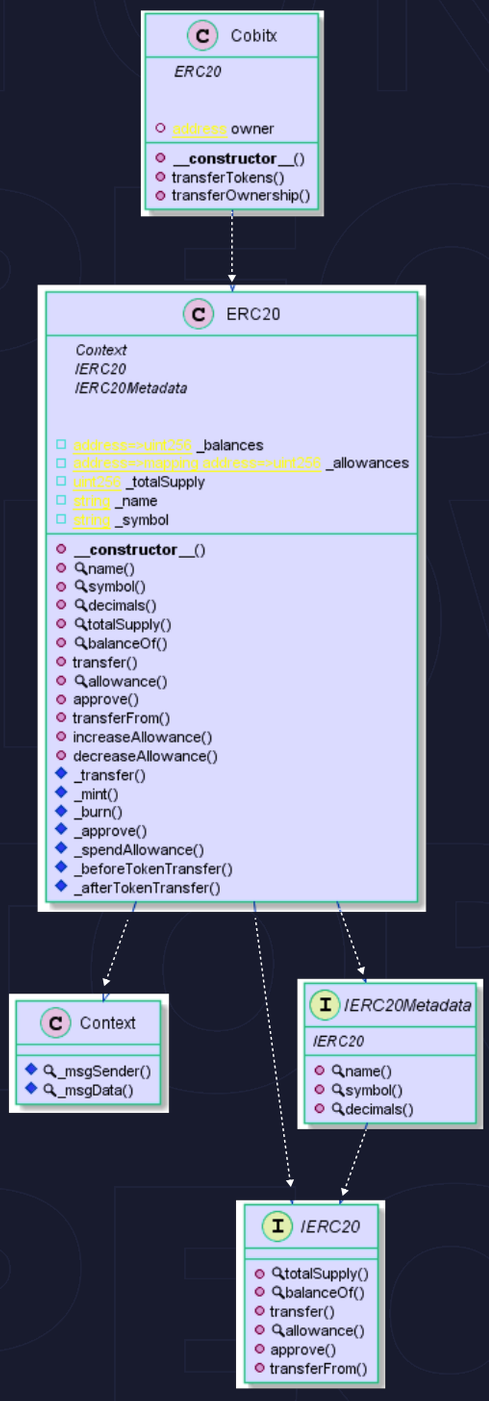
Audit report contains all found security vulnerabilities and other issues in the reviewed code.
Security state of the reviewed smart contract, based on standard audit procedure scope, is "**Secured**".

LOVELY INSPECTOR

ADDENDUM

Code Flow Diagram

Arkham Token



LOVELY INSPECTOR

SECURITY ASSESSMENT REPORT

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> Cobitx.sol

```
INFO:Detectors:
Cobitx.constructor().totalSupply (Cobitx.sol#534) shadows:
        - ERC20.totalSupply() (Cobitx.sol#242-244) (function)
        - IERC20.totalSupply() (Cobitx.sol#65) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Cobitx.transferOwnership(address) (Cobitx.sol#551-554) should emit an event for:
        - owner = newOwner (Cobitx.sol#553)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control
INFO:Detectors:
Context._msgData() (Cobitx.sol#32-34) is never used and should be removed
ERC20._burn(address,uint256) (Cobitx.sol#425-441) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (Cobitx.sol#15) allows old versions
Pragma version^0.8.0 (Cobitx.sol#42) allows old versions
Pragma version^0.8.0 (Cobitx.sol#123) allows old versions
Pragma version^0.8.0 (Cobitx.sol#153) allows old versions
Pragma version^0.8.0 (Cobitx.sol#518) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

INFO:Detectors:
Cobitx.constructor() (Cobitx.sol#529-536) uses literals with too many digits:
        - totalSupply = 990000000 * 10 ** 18 (Cobitx.sol#534)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Slither:Cobitx.sol analyzed (5 contracts with 93 detectors), 11 result(s) found
```

LOVELY INSPECTOR

SOLIDITY STATIC ANALYSIS

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

Cobitx.sol

Gas costs:

Gas requirement of function Cobitx.transferTokens is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 539:4:

Constant/View/Pure functions:

ERC20._afterTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 512:4:

Similar variable names:

ERC20.(string,string) : Variables have very similar names "_name" and "name_".
Note: Modifiers are currently not considered by this static analysis.
Pos: 203:8:

LOVELY INSPECTOR

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 552:8:

LOVELY INSPECTOR

COMPLIANCE ANALYSIS

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

Cobitx.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:14
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:41
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:122
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:152
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:201
Error message for require is too long
Pos: 9:347
Error message for require is too long
Pos: 9:370
Error message for require is too long
Pos: 9:371
Error message for require is too long
Pos: 9:376
Error message for require is too long
Pos: 9:425
Error message for require is too long
Pos: 9:430
Error message for require is too long
Pos: 9:456
Error message for require is too long
Pos: 9:457
Code contains empty blocks
Pos: 94:495
Code contains empty blocks
Pos: 93:511
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver requirement
Pos: 1:517
Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
Pos: 5:528
Error message for require is too long
Pos: 9:539
Error message for require is too long
Pos: 9:540
Error message for require is too long
Pos: 9:551
```

LOVELY INSPECTOR

SOFTWARE ANALYSIS RESULT

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

LOVELY INSPECTOR

INSPECTOR LOVELY

INFO

Website: Inspector.lovely.finance

Telegram community: t.me/inspectorlovely

Twitter: twitter.com/InspectorLovely

inspector.lovely.finance