EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: CyberConnect Token
Website: cyber.co
Platform: Binance Smart Chain
Language: Solidity
Date: March 12th, 2024

Table of contents

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the smart contracts of CyberConnect from cyber.co were audited. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on March 12th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- The `CyberToken` contract, built using Solidity, inherits from several foundational contracts to create a comprehensive ERC20 token with additional functionalities like burning, permit signatures, and voting. Below, provide an overview of the key components and functionalities of this contract.
    - **ERC20:** This is the basic token standard that implements the standard ERC20 interface, including methods like `transfer`, `approve`, `transferFrom`, and `allowance`. It handles the basic token mechanics such as balances and allowances.
    - **ERC20Burnable:** This extends the ERC20 functionality by allowing tokens to be burned (destroyed), reducing the total supply. It includes methods for burning tokens held by the caller (`burn`) and burning tokens on behalf of another account (`burnFrom`).
    - **ERC20Permit:** This introduces the EIP-2612 permit function, which allows token approvals to be made via signatures (meta-transactions) instead of requiring an on-chain transaction from the token holder. It includes nonce management and EIP-712 typed data hashing.
    - **ERC20Votes:** This extension enables a token to be used for voting. It keeps track of vote delegations and voting power over time, integrating functionalities for checkpointing and delegation.

- **Ownable:** This is a simple authorization pattern where there is an owner who has exclusive access to specific functions. It provides methods to transfer and renounce ownership, ensuring that only the owner can perform critical actions such as minting new tokens.
- **CyberToken:** The `CyberToken` contract is a comprehensive implementation of an ERC20 token with additional functionalities for burning tokens, using permit signatures for approvals, and enabling vote delegation and tracking. It utilizes a modular approach by inheriting and combining functionalities from multiple abstract contracts, ensuring code reusability and modularity. The use of `Ownable` ensures that certain critical functions are restricted to the owner, maintaining security and control over the token's life cycle operations.

Audit scope

Name	**Code Review and Security Analysis Report for CyberConnect Token Smart Contract**
Platform	**Binance Smart Chain**
Language	**Solidity**
File	CyberToken.sol
Smart Contract Code	0x14778860E937f509e651192a90589dE711Fb88a9
Audit Date	March 12th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: CyberConnect • Symbol: CYBER • Decimals: 18	**YES, This is valid.**
Ownership control: • Only the owner's address has permission to mint a token. • The current owner can transfer the ownership. • The owner can renounce ownership.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

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, Solhint, 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. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium 1 low, and 2 very low level issues.

Investor Advice: A 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.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Moderated
	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	Moderated
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	Moderated
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Overall Audit Result: PASSED

Business Risk Analysis

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

Overall Audit Result: PASSED

Code Quality

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

The libraries in CyberConnect 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 CyberConnect Token.

The EtherAuthority team has no 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 CyberConnect 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.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	mint	write	The owner can mint unlimited tokens, Centralization Risks	Refer Audit Findings
3	_mint	internal	Passed	No Issue
4	_burn	internal	Passed	No Issue
5	_afterTokenTransfer	internal	Passed	No Issue
6	checkpoints	read	Passed	No Issue
7	numCheckpoints	read	Passed	No Issue
8	delegates	read	Passed	No Issue
9	getVotes	read	Passed	No Issue
10	getPastVotes	read	Passed	No Issue
11	getPastTotalSupply	read	Passed	No Issue
12	_checkpointsLookup	read	Passed	No Issue
13	delegate	write	Passed	No Issue
14	delegateBySig	write	Passed	No Issue
15	_maxSupply	internal	Passed	No Issue
16	_mint	internal	Passed	No Issue
17	_burn	internal	Passed	No Issue
18	_afterTokenTransfer	internal	Passed	No Issue
19	_delegate	internal	Passed	No Issue
20	_moveVotingPower	write	Passed	No Issue
21	_writeCheckpoint	write	Passed	No Issue
22	_add	write	Passed	No Issue
23	_subtract	write	Passed	No Issue
24	_unsafeAccess	write	Passed	No Issue
25	burn	write	Passed	No Issue
26	burnFrom	write	Passed	No Issue
27	permit	write	Passed	No Issue
28	nonces	read	Passed	No Issue
29	DOMAIN_SEPARATOR	external	Passed	No Issue
30	_useNonce	internal	Passed	No Issue
31	name	read	Passed	No Issue
32	symbol	read	Passed	No Issue
33	decimals	read	Passed	No Issue
34	totalSupply	read	Passed	No Issue
35	balanceOf	read	Passed	No Issue
36	transfer	write	Passed	No Issue
37	allowance	read	Passed	No Issue
38	approve	write	Passed	No Issue
39	transferFrom	write	Passed	No Issue

40	increaseAllowance	write	Passed	No Issue
41	decreaseAllowance	write	Passed	No Issue
42	_transfer	internal	Passed	No Issue
43	_mint	internal	Passed	No Issue
44	_burn	internal	Passed	No Issue
45	_approve	internal	Passed	No Issue
46	_spendAllowance	internal	Passed	No Issue
47	_beforeTokenTransfer	internal	Passed	No Issue
48	_afterTokenTransfer	internal	Passed	No Issue
49	onlyOwner	modifier	Passed	No Issue
50	owner	read	Passed	No Issue
51	_checkOwner	internal	Passed	No Issue
52	renounceOwnership	write	Centralization Risks	Refer Audit Findings
53	transferOwnership	write	Centralization Risks	Refer Audit Findings
54	_transferOwnership	internal	Passed	No Issue
55	_domainSeparatorV4	internal	Passed	No Issue
56	_buildDomainSeparator	write	Passed	No Issue
57	_hashTypedDataV4	internal	Passed	No Issue

Severity Definitions

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
Medium	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.

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

(1) The owner can mint unlimited tokens: **CyberToken.sol**

There is no limit for minting CYBER tokens. Thus the owner can mint unlimited tokens to any account.

Resolution: There should be a limit for minting or need to confirm, if it is a part of the plan then disregard this issue.

Very Low / Informational / Best practices:

(1) Use the latest solidity version: **CyberToken.sol**

Use the latest solidity version while contract deployment to prevent any compiler version-level bugs.

Resolution: Please use 0.8.24 which is the latest version.

(2) Centralization Risks:

CyberToken.sol

- In the contract, onlyOwner can mint a token.

Ownable.sol

In the contract onlyOwner as an owner has authority on the following function:

- renounceOwnership()
- transferOwnership()

Resolution: We suggest carefully managing the onlyOwner private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practice.

Centralization

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

CyberToken.sol

- mint: Mint a new token by the owner.

Ownable.sol

- renounce Ownership: Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- transferOwnership: The current owner can transfer ownership of the contract to a new account.

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

Conclusion

We were given a contract code in the form of Etherscan web links. 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. but those are not critical. 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.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: 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).

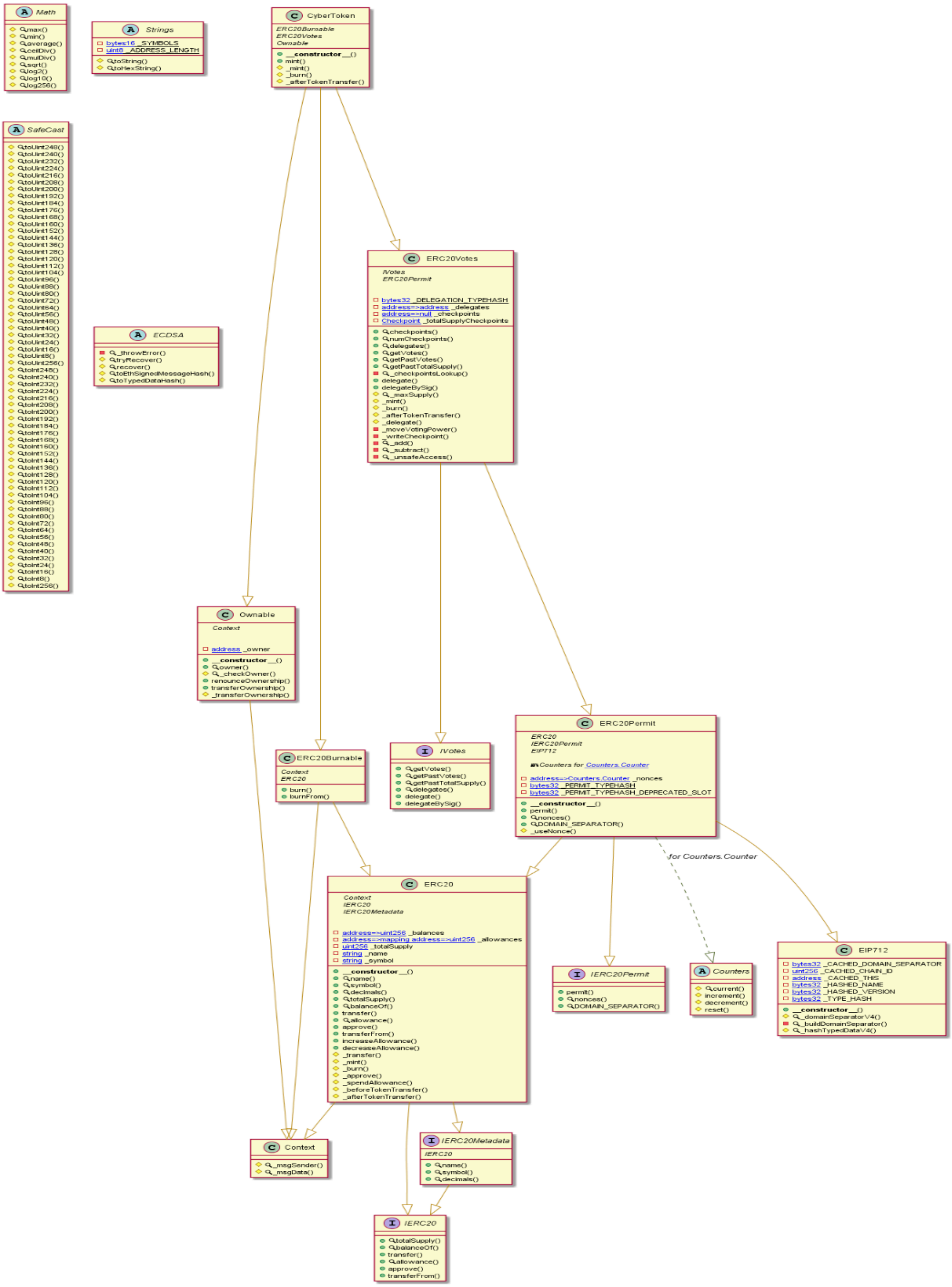
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While 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 also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

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

Appendix

Code Flow Diagram - CyberConnect Token



Email: audit@EtherAuthority.io

Slither Results Log

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 >> CyberToken.sol

```
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (CyberToken.sol#231-311) has bitwise-xor operator ^ instead of the exponentiation operator **:
         - inverse = (3 * denominator) ^ 2 (CyberToken.sol#293)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-exponentiation
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (CyberToken.sol#231-311) performs a multiplication on the result of a division:
        - denominator = denominator / twos (CyberToken.sol#278)
        - inverse = (3 * denominator) ^ 2 (CyberToken.sol#293)
Math.mulDiv(uint256,uint256,uint256) (CyberToken.sol#231-311) performs a multiplication on the result of a division:
        - denominator = denominator / twos (CyberToken.sol#278)
        - inverse *= 2 - denominator * inverse (CyberToken.sol#297)
Math.mulDiv(uint256,uint256,uint256) (CyberToken.sol#231-311) performs a multiplication on the result of a division:
        - denominator = denominator / twos (CyberToken.sol#278)
        - inverse *= 2 - denominator * inverse (CyberToken.sol#298)
Math.mulDiv(uint256,uint256,uint256) (CyberToken.sol#231-311) performs a multiplication on the result of a division:
        - denominator = denominator / twos (CyberToken.sol#278)
        - inverse *= 2 - denominator * inverse (CyberToken.sol#299)
Math.mulDiv(uint256,uint256,uint256) (CyberToken.sol#231-311) performs a multiplication on the result of a division:
        - denominator = denominator / twos (CyberToken.sol#278)
        - inverse *= 2 - denominator * inverse (CyberToken.sol#300)
Math.mulDiv(uint256,uint256,uint256) (CyberToken.sol#231-311) performs a multiplication on the result of a division:
        - denominator = denominator / twos (CyberToken.sol#278)
        - inverse *= 2 - denominator * inverse (CyberToken.sol#301)
Math.mulDiv(uint256,uint256,uint256) (CyberToken.sol#231-311) performs a multiplication on the result of a division:
        - denominator = denominator / twos (CyberToken.sol#278)
        - inverse *= 2 - denominator * inverse (CyberToken.sol#302)
Math.mulDiv(uint256,uint256,uint256) (CyberToken.sol#231-311) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (CyberToken.sol#281)
        - result = prod0 * inverse (CyberToken.sol#308)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ERC20Votes._writeCheckpoint(ERC20Votes.Checkpoint[],function(uint256,uint256) returns(uint256),uint256) (CyberToken.sol#2780-2797) uses a dangerous strict equality:
        - pos > 0 && oldCkpt.fromBlock == block.number (CyberToken.sol#2792)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
ERC20Permit.constructor(string).name (CyberToken.sol#2487) shadows:
        - ERC20.name() (CyberToken.sol#2136-2138) (function)
        - IERC20Metadata.name() (CyberToken.sol#90) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (CyberToken.sol#2492-2511) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (CyberToken.sol#2501)
```

```
ERC20Votes.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (CyberToken.sol#2687-2704) uses
timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,ERC20Votes: signature expired) (CyberToken.sol#
2695)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Math.mulDiv(uint256,uint256,uint256) (CyberToken.sol#231-311) uses assembly
        - INLINE ASM (CyberToken.sol#242-246)
        - INLINE ASM (CyberToken.sol#262-269)
        - INLINE ASM (CyberToken.sol#276-285)
Strings.toString(uint256) (CyberToken.sol#530-550) uses assembly
        - INLINE ASM (CyberToken.sol#536-538)
        - INLINE ASM (CyberToken.sol#542-544)
ECDSA.tryRecover(bytes32,bytes) (CyberToken.sol#1795-1812) uses assembly
        - INLINE ASM (CyberToken.sol#1803-1807)
ERC20Votes._unsafeAccess(ERC20Votes.Checkpoint[],uint256) (CyberToken.sol#2810-2815) uses assembly
        - INLINE ASM (CyberToken.sol#2811-2814)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
INFO:Detectors:
Pragma version^0.8.13 (CyberToken.sol#4) allows old versions
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function IERC20Permit.DOMAIN_SEPARATOR() (CyberToken.sol#149) is not in mixedCase
Variable EIP712._CACHED_DOMAIN_SEPARATOR (CyberToken.sol#1958) is not in mixedCase
Variable EIP712._CACHED_CHAIN_ID (CyberToken.sol#1959) is not in mixedCase
Variable EIP712._CACHED_THIS (CyberToken.sol#1960) is not in mixedCase
Variable EIP712._HASHED_NAME (CyberToken.sol#1962) is not in mixedCase
Variable EIP712._HASHED_VERSION (CyberToken.sol#1963) is not in mixedCase
Variable EIP712._TYPE_HASH (CyberToken.sol#1964) is not in mixedCase
Function ERC20Permit.DOMAIN_SEPARATOR() (CyberToken.sol#2524-2526) is not in mixedCase
Variable ERC20Permit._PERMIT_TYPEHASH_DEPRECATED_SLOT (CyberToken.sol#2480) is not in mixedCase
Parameter CyberToken.mint(address,uint256)._account (CyberToken.sol#2839) is not in mixedCase
Parameter CyberToken.mint(address,uint256)._amount (CyberToken.sol#2839) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-
conventions
INFO:Slither:CyberToken.sol analyzed (17 contracts with 93 detectors), 117 result(s) found
```

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.

CyberToken.sol

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
more
Pos: 2811:15:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 2501:23:

Gas costs:

Gas requirement of function CyberToken.delegateBySig 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: 2687:11:

Gas costs:

Gas requirement of function CyberToken.mint 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: 2839:11:

Similar variable names:

CyberToken._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 2858:36:

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: 2718:15:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 363:35:

Solhint Linter

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.

CyberToken.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:3
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:241
Provide an error message for require
Pos: 13:253
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:261
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:275
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:535
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 17:541
Error message for require is too long
Pos: 9:650
Error message for require is too long
Pos: 9:1747
Error message for revert is too long
Pos: 13:1770
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 13:1802
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:1979
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:2051
Error message for require is too long
Pos: 9:2093
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:2127
Error message for require is too long
Pos: 9:2277
Error message for require is too long
Pos: 9:2304
Error message for require is too long
Pos: 9:2305
Error message for require is too long
Pos: 9:2310
Error message for require is too long
```

```
Pos: 9:2359
Error message for require is too long
Pos: 9:2364
Error message for require is too long
Pos: 9:2394
Error message for require is too long
Pos: 9:2395
Code contains empty blocks
Pos: 24:2441
Code contains empty blocks
Pos: 24:2461
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:2486
Code contains empty blocks
Pos: 55:2486
Avoid making time-based decisions in your business logic
Pos: 17:2500
Avoid making time-based decisions in your business logic
Pos: 17:2694
Error message for require is too long
Pos: 9:2717
Avoid using inline assembly. It is acceptable only in rare cases
Pos: 9:2810
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:2832
Code contains empty blocks
Pos: 78:2832
```

Software analysis result:

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

Email: audit@EtherAuthority.io