EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

# SMART CONTRACT

## Security Audit Report

Project: GCS Token
WebSite: www.goldchainex.com
Platform: Binance Smart Chain
Language: Solidity
Date: February 23rd, 2022

# 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

EtherAuthority was contracted by the Goldchainex.ou team to perform the Security audit of the GCS Token smart contract code. 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 February 23rd, 2022.

**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

GCS is a standard BEP20 token smart contract. This audit only considers GCS token smart contract, and does not cover any other smart contracts on the platform.

## Audit scope

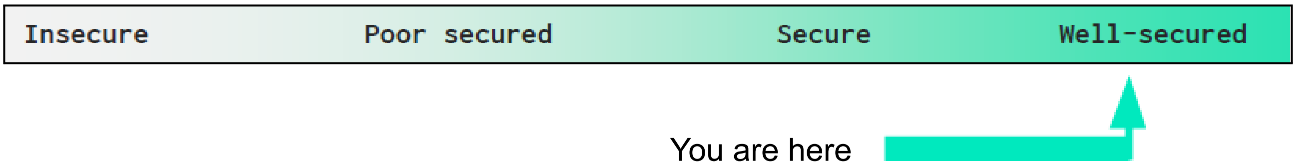
| **Name** | **Code Review and Security Analysis Report for GCS Token Smart Contract** |
|---|---|
| **Platform** | **BSC / Solidity** |
| **File** | GCS.sol |
| **File MD5 Hash** | CCE3AF75661A2E376560BFBE8B234254 |
| **Updated File MD5 Hash** | 4514D8FAFDD63F78969E437ADD9DE7B3 |
| **Online code** | 0x79bfce8b9419a9d2221df9f2898d1d4e24353e8e |
| **Updated Online code** | 0x3d2bb1f7ab5d64C3917DbE03d37843421A42e0cD |
| **Audit Date** | February 23rd, 2022 |
| **Revise Audit Date** | February 28th, 2022 |

## Claimed Smart Contract Features

| Claimed Feature Detail | Our Observation |
| --- | --- |
| **Tokenomics:**<br>● Name: GCS<br>● Symbol: GCS<br>● Decimals: 18<br>● Total Supply: 10 Million<br>● Max Supply: 10 Million | **YES, This is valid.** |

# Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Well Secured"**. This token contract does contain owner control, which does not make it fully decentralized.




We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on 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 0 low and some very low level issues. These issues are not critical ones. All the issues have been resolved.**

**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.

## Technical Quick Stats

| Main Category | Subcategory | Result |
|---|---|---|
| Contract Programming | Solidity version not specified | Passed |
| | Solidity version too old | Passed |
| | Integer overflow/underflow | Passed |
| | Function input parameters lack of 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 not set | Passed |
| | "Short Address" Attack | Passed |
| | "Double Spend" Attack | Passed |

**Overall Audit Result: PASSED**

## Code Quality

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

The libraries in GCS 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 GCS Token.

The GCS Token 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 **not well** commented on smart contracts.

## Documentation

We were given a GCS Token smart contracts code in the form of a BSCScan web link.The hash of that code is mentioned above in the table.

As mentioned above, code parts are **not well** commented. So it is not 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.

Another source of information was its official website [https://www.goldchainex.com/](https://www.goldchainex.com/) which provided rich information about the project architecture and tokenomics.

## 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 used in external smart contract calls.

# AS-IS overview

### Functions

| Sl. | Functions | Type | Obser-vation | Conclusion |
|---|---|---|---|---|
| 1 | constructor | write | Passed | No Issue |
| 2 | mint | write | access only Owner | No Issue |
| 3 | burn | write | Passed | No Issue |
| 4 | owner | read | Passed | No Issue |
| 5 | onlyOwner | modifier | Passed | No Issue |
| 6 | renounceOwnership | write | access only Owner | No Issue |
| 7 | transferOwnership | write | access only Owner | No Issue |
| 8 | _transferOwnership | internal | Passed | No Issue |
| 9 | name | read | Passed | No Issue |
| 10 | symbol | read | Passed | No Issue |
| 11 | decimals | read | Passed | No Issue |
| 12 | totalSupply | read | Passed | No Issue |
| 13 | balanceOf | read | Passed | No Issue |
| 14 | transfer | write | Passed | No Issue |
| 15 | allowance | read | Passed | No Issue |
| 16 | approve | write | Passed | No Issue |
| 17 | transferFrom | write | Passed | No Issue |
| 18 | increaseAllowance | write | Passed | No Issue |
| 19 | decreaseAllowance | write | Passed | No Issue |
| 20 | _transfer | internal | Passed | No Issue |
| 21 | _mint | internal | Passed | No Issue |
| 22 | _burn | internal | Passed | No Issue |
| 23 | _approve | internal | Passed | No Issue |
| 24 | _beforeTokenTransfer | internal | Passed | No Issue |
| 25 | _afterTokenTransfer | 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

No Low severity vulnerabilities were found.

## Very Low / Informational / Best practices:

(1) Owner can mint unlimited tokens:

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

**Resolution:** There should be a limit for minting for better tokenomics.

**Status:** Fixed

(2) Owner can burn anyone's token:

Owner can burn any users' tokens.

**Resolution:** We suggest changing the code so only token holders can burn their own tokens and not anyone else. Not even a contract creator.

**Status:** Fixed

# Centralization

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

- mint: The Owner can mint the amount to the account.

## Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We have not observed any major issues in the smart contracts. So, **it's good to go to 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 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 contract, based on standard audit procedure scope, is **"Well 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 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, 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).

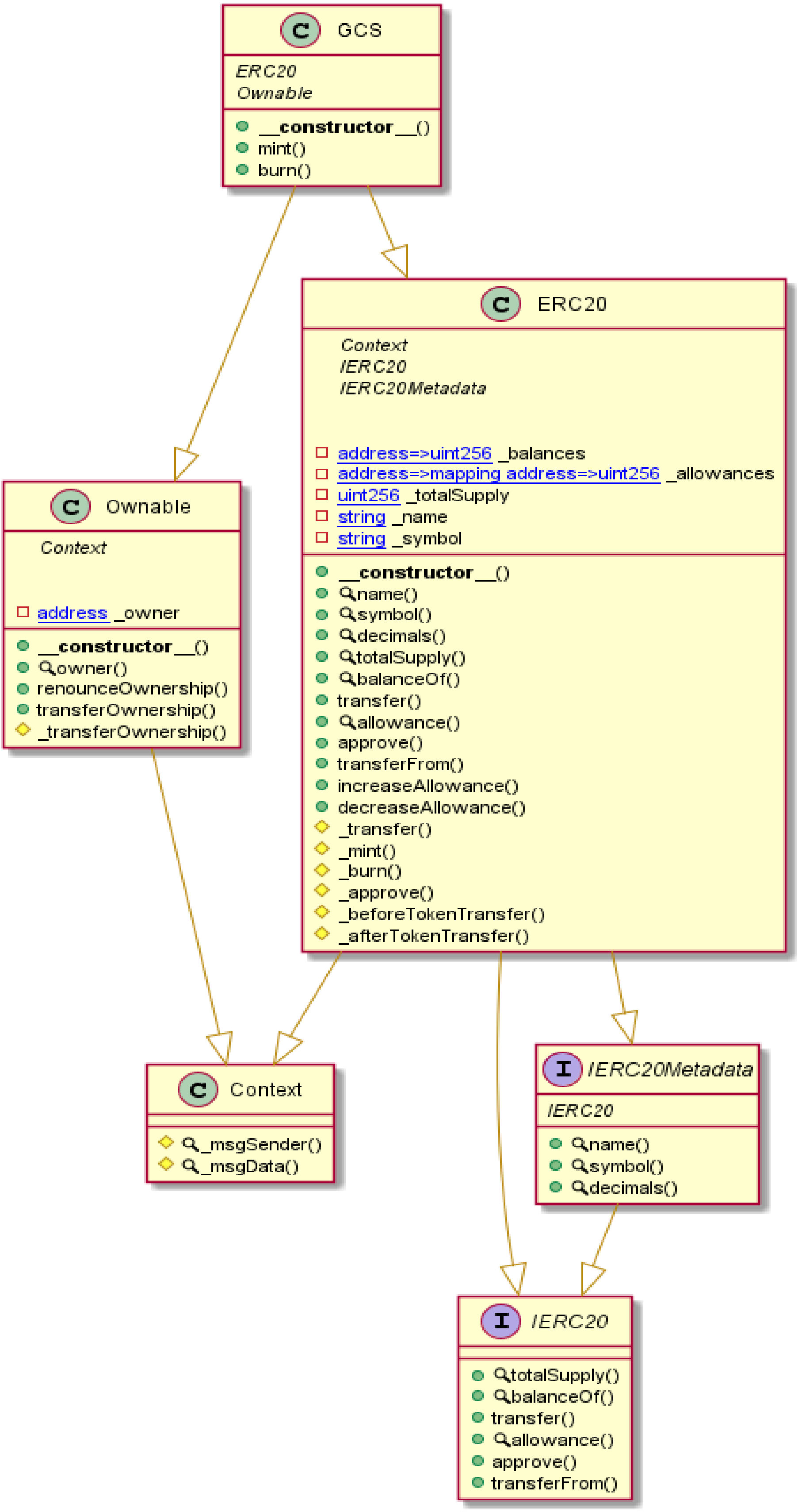
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 explicit security of the audited smart contracts.

# Appendix

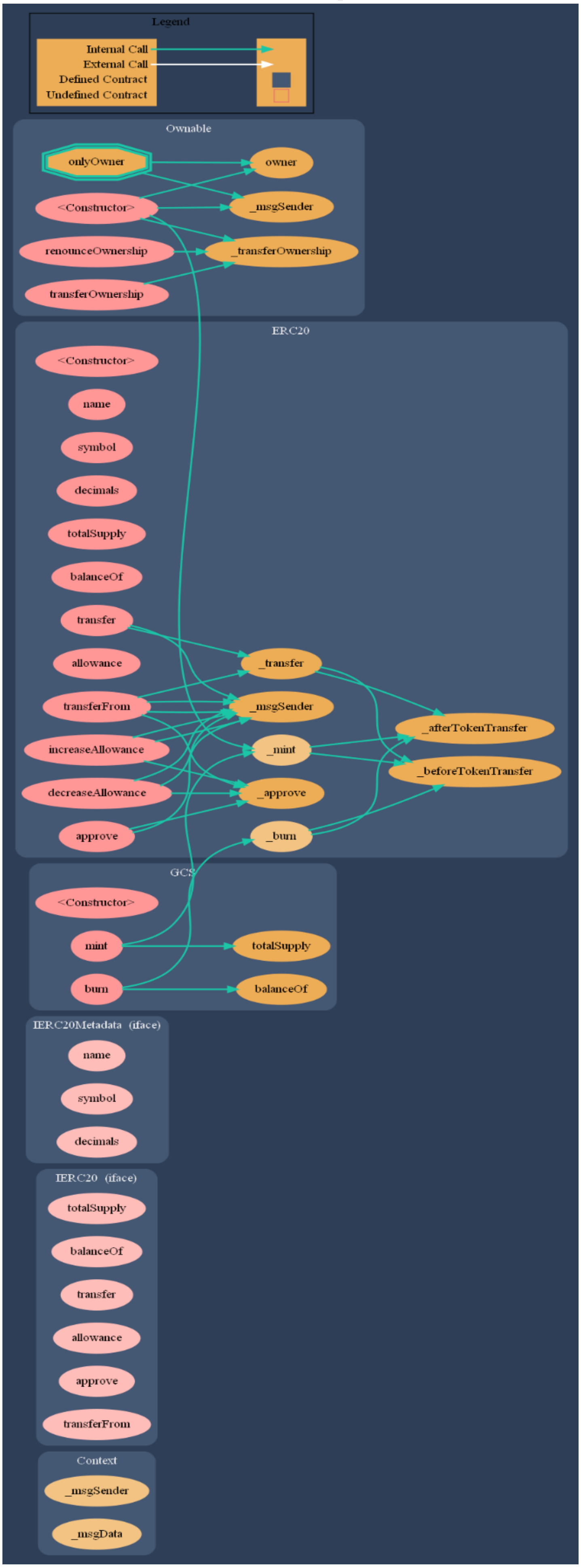
## Unified Modeling Language (UML) - GCS Token





**Email: audit@EtherAuthority.io**

# Call Graph



Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
_transfer
allowance
transferFrom
_msgSender
_afterTokenTransfer
increaseAllowance
_mint
_beforeTokenTransfer
decreaseAllowance
_approve
approve
_burn
GCS
<Constructor>
mint
totalSupply
burn
balanceOf
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData

# Automatic general report

Files Description Table

| File Name | SHA-1 Hash |

|-------------|--------------|

| GCS.sol | 84f0e8fb0bee70e6ce7cb9839393c0cde6cdd54c |

|

Contracts Description Table

| Contract | Type | Bases | | |

|:----------:|:-------------------:|:----------------:|:----------------:|:---------------:|

| └ | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |

||||||

| **Context** | Implementation | |||

| └ | _msgSender | Internal 🔒 | | |

| └ | _msgData | Internal 🔒 | | |

||||||

| **IERC20** | Interface | |||

| └ | totalSupply | External ❗️ | |NO❗️ |

| └ | balanceOf | External ❗️ | |NO❗️ |

| └ | transfer | External ❗️ | 🛑 |NO❗️ |

| └ | allowance | External ❗️ | |NO❗️ |

| └ | approve | External ❗️ | 🛑 |NO❗️ |

| └ | transferFrom | External ❗️ | 🛑 |NO❗️ |

||||||

| **IERC20Metadata** | Interface | IERC20 |||

| └ | name | External ❗️ | |NO❗️ |

| └ | symbol | External ❗️ | |NO❗️ |

| └ | decimals | External ❗️ | |NO❗️ |

||||||

| **Ownable** | Implementation | Context |||

| └ | <Constructor> | Public ❗️ | 🛑 |NO❗️ |

| └ | owner | Public ❗️ | |NO❗️ |

| └ | renounceOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | transferOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | _transferOwnership | Internal 🔒 | 🛑 | |

||||||

| **ERC20** | Implementation | Context, IERC20, IERC20Metadata |||

| └ | <Constructor> | Public ❗ | 🛑 |NO ❗ |

| └ | name | Public ❗ | |NO ❗ |

| └ | symbol | Public ❗ | |NO ❗ |

| └ | decimals | Public ❗ | |NO ❗ |

| └ | totalSupply | Public ❗ | |NO ❗ |

| └ | balanceOf | Public ❗ | |NO ❗ |

| └ | transfer | Public ❗ | 🛑 |NO ❗ |

| └ | allowance | Public ❗ | |NO ❗ |

| └ | approve | Public ❗ | 🛑 |NO ❗ |

| └ | transferFrom | Public ❗ | 🛑 |NO ❗ |

| └ | increaseAllowance | Public ❗ | 🛑 |NO ❗ |

| └ | decreaseAllowance | Public ❗ | 🛑 |NO ❗ |

| └ | _transfer | Internal 🔒 | 🛑 | |

| └ | _mint | Internal 🔒 | 🛑 | |

| └ | _burn | Internal 🔒 | 🛑 | |

| └ | _approve | Internal 🔒 | 🛑 | |

| └ | _beforeTokenTransfer | Internal 🔒 | 🛑 | |

| └ | _afterTokenTransfer | Internal 🔒 | 🛑 | |

||||||

| **GCS** | Implementation | ERC20, Ownable |||

| └ | <Constructor> | Public ❗ | 🛑 | ERC20 |

| └ | mint | Public ❗ | 🛑 | onlyOwner |

| └ | burn | Public ❗ | 🛑 | |

||||||

Legend

| Symbol | Meaning |

|:--------:|-----------|

| 🛑 | Function can modify state |

| 💵 | Function is payable |

# Slither Results Log

## Slither log >> GCS.sol

```
INFO:Detectors:
Context._msgData() (GCS.sol#25-27) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.4 (GCS.sol#8) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter GCS.mint(address,uint256)._to (GCS.sol#533) is not in mixedCase
Parameter GCS.mint(address,uint256)._amount (GCS.sol#533) is not in mixedCase
Parameter GCS.burn(uint256)._amount (GCS.sol#538) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (GCS.sol#179-181)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (GCS.sol#187-190)
name() should be declared external:
        - ERC20.name() (GCS.sol#231-233)
symbol() should be declared external:
        - ERC20.symbol() (GCS.sol#239-241)
decimals() should be declared external:
        - ERC20.decimals() (GCS.sol#256-258)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (GCS.sol#282-285)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (GCS.sol#290-292)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (GCS.sol#301-304)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (GCS.sol#319-333)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (GCS.sol#347-350)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (GCS.sol#366-374)
mint(address,uint256) should be declared external:
        - GCS.mint(address,uint256) (GCS.sol#533-536)
```

```
mint(address,uint256) should be declared external:
        - GCS.mint(address,uint256) (GCS.sol#533-536)
burn(uint256) should be declared external:
        - GCS.burn(uint256) (GCS.sol#538-541)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:GCS.sol analyzed (6 contracts with 75 detectors), 19 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

# Solidity Static Analysis

## GCS.sol

Gas & Economy

**Gas costs:**

Gas requirement of function ERC20.name 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: 231:4:

**Gas costs:**

Gas requirement of function GCS.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: 533:4:

**Gas costs:**

Gas requirement of function GCS.burn 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: 538:4:

Miscellaneous

**Constant/View/Pure functions:**

IERC20.transfer(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 52:4:

**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: 520:4:

**Similar variable names:**

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 450:34:

**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: 450:8:

**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: 479:8:

**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: 480:8:

**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: 534:8:

**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: 539:8:


**Email: audit@EtherAuthority.io**

## Solhint Linter

**GCS.sol**

```
GCS.sol:328:18: Error: Parse error: missing ';' at '{'
GCS.sol:369:18: Error: Parse error: missing ';' at '{'
GCS.sol:402:18: Error: Parse error: missing ';' at '{'
GCS.sol:451:18: Error: Parse error: missing ';' at '{'
```

**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**