Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	CGI Token
Platform:	Ethereum
Language:	Solidity
Date:	June 25th, 2022

Table of contents

Email: audit@EtherAuthority.io

`

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.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the CGI team to perform the Security audit of the 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 June 25th, 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

- The CGI Token is an ERC1155 token which has functionalities like minting, burning, pause/unpause, blacklist addresses, and bridge using messageProxy.
- The CGI Token contract inherits the ERC1155PresetMinterPauser, ERC1155Supply, Strings, ERC1155Holder, ERC2981 standard smart contracts from the OpenZeppelin library.
- These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

Name	**Code Review and Security Analysis Report for CGI Token Smart Contract**
Platform	**Ethereum / Solidity**
File	LiveCGIToken.sol
File Sha1 Hash	6ec900b3f66f68c6fd9cf4fb5ab6555059953df2
Updated File Sha1 Hash	8992389217c9c9c0404dcec1128e5877d34403ec
Audit Date	June 25th, 2022
Revision Date	June 29th, 2022

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Token Name: CGI Token ● ERC1155 and ERC2981 compliance ● This contract has functions like: bridge, post message, Interface support, etc. ● Minting and burning ability ● in-built royalty	**YES, This is valid.**
Ownership Control: ● Owner can add a new address to the blacklist. ● Owner can remove addresses from the blacklist. ● Owner can also set other state variables	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **“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 2 low and some very low level issues.
All issues have been acknowledged / resolved in the revised 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.

Email: audit@EtherAuthority.io

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	Moderated
	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	Moderated
	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**

Email: audit@EtherAuthority.io

Code Quality

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

The libraries in CGI 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 CGI Token.

The CGI 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 in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a CGI Token smart contract code in the form of an Etherscan weblink. The hash of that code is mentioned above in the table.

As mentioned above, code parts are not well commented. so it's not easy to understand its programming logic.

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	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	mint	write	Passed	No Issue
3	mintBatch	write	Passed	No Issue
4	pause	write	Passed	No Issue
5	unpause	write	Passed	No Issue
6	supportsInterface	write	Passed	No Issue
7	_beforeTokenTransfer	write	Passed	No Issue
8	totalSupply	read	Passed	No Issue
9	exists	read	Passed	No Issue
10	_beforeTokenTransfer	internal	Passed	No Issue
11	onERC1155Received	write	Passed	No Issue
12	onERC1155BatchReceived	write	Passed	No Issue
13	supportsInterface	read	Passed	No Issue
14	royaltyInfo	external	Passed	No Issue
15	_feeDenominator	internal	Passed	No Issue
16	_setDefaultRoyalty	internal	Passed	No Issue
17	_deleteDefaultRoyalty	internal	Passed	No Issue
18	_setTokenRoyalty	internal	Passed	No Issue
19	_resetTokenRoyalty	internal	Passed	No Issue
20	addBlacklist	external	Passed	No Issue
21	removeBlacklist	external	Passed	No Issue
22	uri	read	Passed	No Issue
23	setURI	external	Passed	No Issue
24	setMessageProxy	external	Passed	No Issue
25	setTargetChainHash	external	Passed	No Issue
26	setTargetContract	external	Passed	No Issue
27	bridge	external	Passed	No Issue
28	postMessage	external	Passed	No Issue
29	_mint	internal	Passed	No Issue
30	_mintBatch	internal	High gas consuming loops, Infinite loops possibility	Refer Audit Findings
31	_burn	internal	Passed	No Issue
32	_burnBatch	internal	High gas consuming loops, Infinite loops possibility	Refer Audit Findings
33	setTokenRoyalty	external	Critical operation lacks event log	Refer Audit Findings
34	resetTokenRoyalty	internal	Critical operation lacks event log	Refer Audit Findings
35	setRoyaltyfeeNumerator	external	Critical operation lacks event log	Refer Audit Findings

36	supportsInterface	read	Passed	No Issue
37	_beforeTokenTransfer	internal	Passed	No Issue
38	whenNotPaused	modifier	Compile time error	Refer Audit Findings

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.

Email: audit@EtherAuthority.io

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) High gas consuming loops:

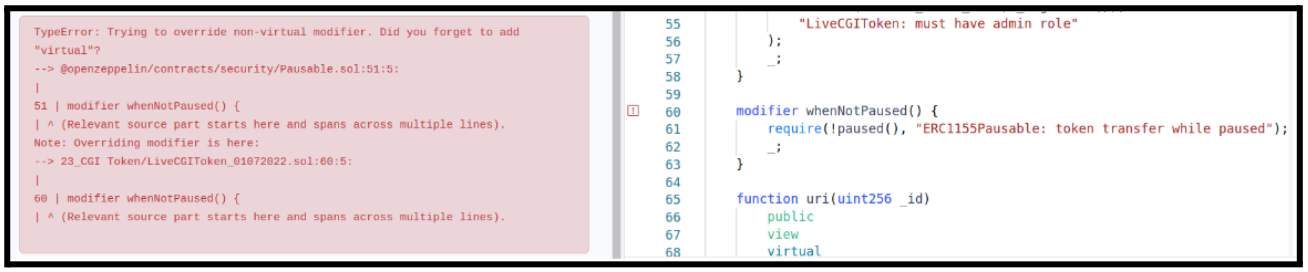
```
function _mintBatch(
    address to,
    uint256[] memory ids,
    uint256[] memory amounts,
    bytes memory data
) internal virtual override {
    for (uint256 i = 0; i < ids.length; i++) {
        if (!exists(ids[i]))
            _setTokenRoyalty(ids[i], to, royaltyfeeNumerator);
    }

    super._mintBatch(to, ids, amounts, data);
}
```

The function _mintBatch has an unbound loop. This does not create a major security or logical vulnerability, but it may hit the block's gas limit if there are high numbers of entries used in the loop. This is true for the _burnBatch function as well.

Resolution: the best practice is to set a limit on the number of entries that are expected. On another hand, this can be safely acknowledged that only a limited number of tokens will be minted in a batch.

Status: Acknowledged.

(2) Compile time error:



Overriding modifier is missing the "override" specifier.

Resolution: We suggest removing the whenNotPaused() modifier to avoid this error. This modifier is already defined in the Pausable contract.

Very Low / Informational / Best practices:

(1) Unlocked Compiler Version:

The contract uses the "^" prefix specifier, using the Unlocked compiler version. Unlocked compiler version code of the smart contract, and that gives permission to the users to compile it one higher than a particular version.

Resolution: We suggest that the compiler version is unlocked instead of the locked compiler version. The following line of code can be added to the project:

pragma solidity 0.8.10;

Status: Acknowledged.

(2) Critical operation lacks event log:

Missing event log for:

- setRoyaltyfeeNumerator
- resetTokenRoyalty
- setRoyaltyfeeNumerator

Resolution: Please write an event log for listed events.

Status: Acknowledged.

(3) Double check the name of the token

The token tracker in the block explorer shows ERC1155. Just to double confirm that it should appear as a CGI Token, or your main brand keyword.

Status: Fixed.

(4) External contracts are used, which are not in the audit scope.

This smart contract uses external contract addresses, which are _messageProxy and targetContract. These smart contracts are not part of this audit scope and thus not audited. The owner should ensure that those contracts are safe and audited.

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:

- addBlacklist: Admin can add a new address in the blacklist.
- removeBlacklist: Admin can remove address from the blacklist.
- setMessageProxy: Admin can set a message proxy address.
- setTargetChainHash: Admin can set target chain hash code.
- setTargetContract: Admin can set target contract address.
- setTokenRoyalty: Admin can set token royalty id.
- resetTokenRoyalty: Admin can reset token royalty id.
- setRoyaltyfeeNumerator: Admin can set royalty fee numerator.

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

Conclusion

We were given a contract code in the form of an Etherscan weblink. And we have used all possible tests based on given objects as files. We have not observed any major issues. 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 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".**

Email: audit@EtherAuthority.io

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).

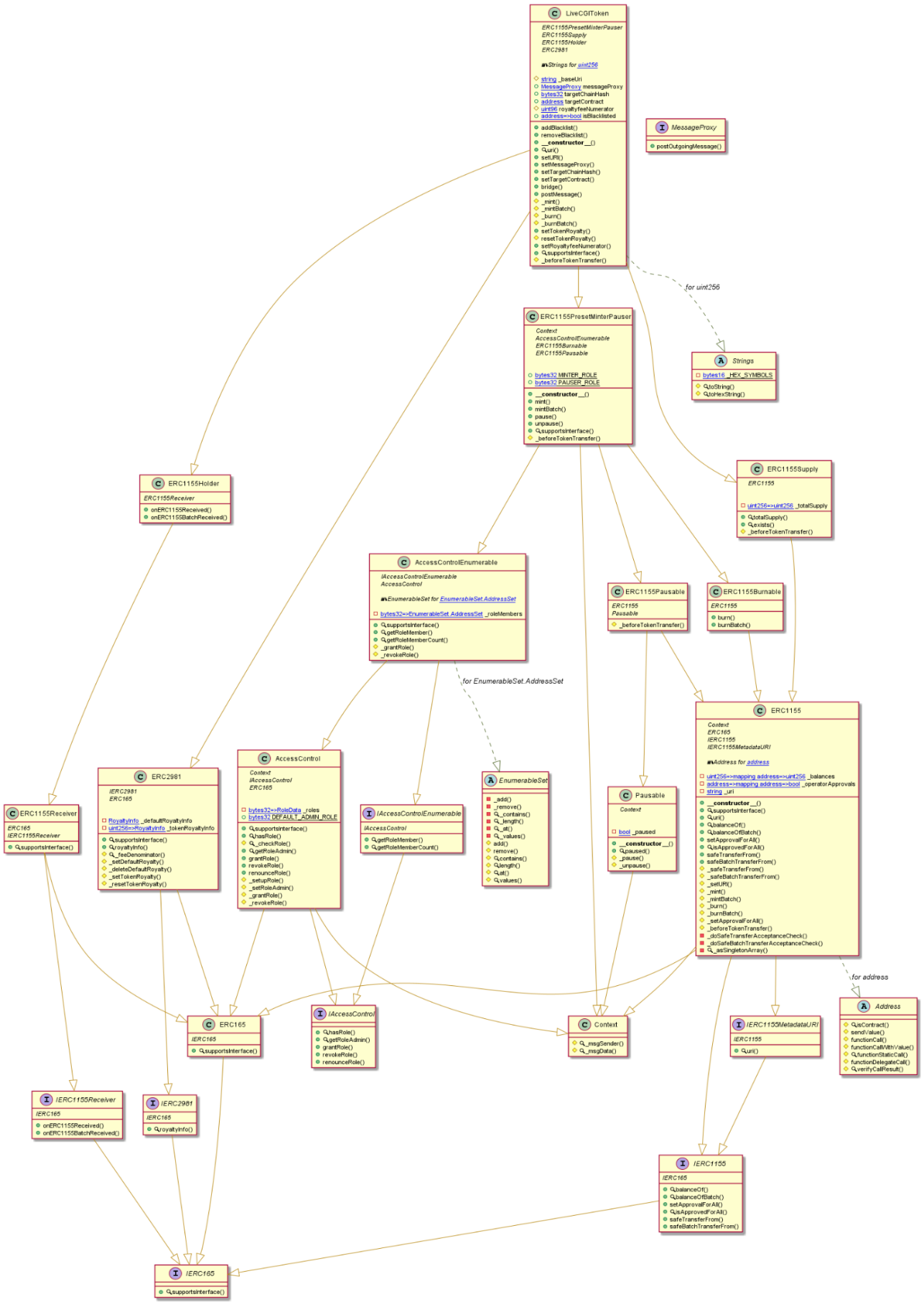
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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

Code Flow Diagram - CGI Token



Slither Results Log

Slither Log >> LiveCGIToken.sol

```
INFO:Detectors:
ERC1155PresetMinterPauser.constructor(string).uri (LiveCGIToken.sol#1082) shadows:
        - ERC1155.uri(uint256) (LiveCGIToken.sol#645-647) (function)
        - IERC1155MetadataURI.uri(uint256) (LiveCGIToken.sol#413) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
LiveCGIToken.setTargetContract(address)._targetContract (LiveCGIToken.sol#1237) lacks a zero-check on :
                - targetContract = _targetContract (LiveCGIToken.sol#1242)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).response (LiveCGIToken.sol#882
in ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (LiveCGIToken.sol#873-892) potentiall
sed before declaration: response != IERC1155Receiver.onERC1155Received.selector (LiveCGIToken.sol#883)
Variable 'ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes).reason (LiveCGIToken.sol#886)'
 ERC1155._doSafeTransferAcceptanceCheck(address,address,address,uint256,uint256,bytes) (LiveCGIToken.sol#873-892) potentially
d before declaration: revert(string)(reason) (LiveCGIToken.sol#887)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).response (LiveCGIToke
ol#904)' in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (LiveCGIToken.sol#8
915) potentially used before declaration: response != IERC1155Receiver.onERC1155BatchReceived.selector (LiveCGIToken.sol#906)
Variable 'ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes).reason (LiveCGIToken.
#909)' in ERC1155._doSafeBatchTransferAcceptanceCheck(address,address,address,uint256[],uint256[],bytes) (LiveCGIToken.sol#894
5) potentially used before declaration: revert(string)(reason) (LiveCGIToken.sol#910)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
EnumerableSet.values(EnumerableSet.AddressSet) (LiveCGIToken.sol#117-126) uses assembly
        - INLINE ASM (LiveCGIToken.sol#121-123)
EnumerableSet.values(EnumerableSet.UintSet) (LiveCGIToken.sol#153-162) uses assembly
        - INLINE ASM (LiveCGIToken.sol#157-159)
Address.verifyCallResult(bool,bytes,string) (LiveCGIToken.sol#307-325) uses assembly
        - INLINE ASM (LiveCGIToken.sol#317-320)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
AccessControl._setRoleAdmin(bytes32,bytes32) (LiveCGIToken.sol#492-496) is never used and should be removed
Address.functionCall(address,bytes) (LiveCGIToken.sol#244-246) is never used and should be removed
Address.functionCall(address,bytes,string) (LiveCGIToken.sol#248-254) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (LiveCGIToken.sol#256-262) is never used and should be removed
```

```
Address.functionCall(address,bytes,string) (LiveCGIToken.sol#248-254) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (LiveCGIToken.sol#256-262) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (LiveCGIToken.sol#264-275) is never used and should be removed
Address.functionDelegateCall(address,bytes) (LiveCGIToken.sol#292-294) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (LiveCGIToken.sol#296-305) is never used and should be removed
Address.functionStaticCall(address,bytes) (LiveCGIToken.sol#277-279) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (LiveCGIToken.sol#281-290) is never used and should be removed
Address.sendValue(address,uint256) (LiveCGIToken.sol#237-242) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (LiveCGIToken.sol#307-325) is never used and should be removed
Context._msgData() (LiveCGIToken.sol#421-423) is never used and should be removed
ERC1155Pausable._beforeTokenTransfer(address,address,address,uint256[],uint256[],bytes) (LiveCGIToken.sol#1030-1041) is never
d and should be removed
ERC1155PresetMinterPauser._beforeTokenTransfer(address,address,address,uint256[],uint256[],bytes) (LiveCGIToken.sol#1131-1140)
 never used and should be removed
ERC2981._deleteDefaultRoyalty() (LiveCGIToken.sol#605-607) is never used and should be removed
ERC2981._setDefaultRoyalty(address,uint96) (LiveCGIToken.sol#598-603) is never used and should be removed
EnumerableSet._values(EnumerableSet.Set) (LiveCGIToken.sol#59-61) is never used and should be removed
EnumerableSet.add(EnumerableSet.Bytes32Set,bytes32) (LiveCGIToken.sol#68-70) is never used and should be removed
EnumerableSet.add(EnumerableSet.UintSet,uint256) (LiveCGIToken.sol#133-135) is never used and should be removed
EnumerableSet.at(EnumerableSet.Bytes32Set,uint256) (LiveCGIToken.sol#84-86) is never used and should be removed
EnumerableSet.at(EnumerableSet.UintSet,uint256) (LiveCGIToken.sol#149-151) is never used and should be removed
EnumerableSet.contains(EnumerableSet.AddressSet,address) (LiveCGIToken.sol#105-107) is never used and should be removed
EnumerableSet.contains(EnumerableSet.Bytes32Set,bytes32) (LiveCGIToken.sol#76-78) is never used and should be removed
EnumerableSet.contains(EnumerableSet.UintSet,uint256) (LiveCGIToken.sol#141-143) is never used and should be removed
EnumerableSet.length(EnumerableSet.Bytes32Set) (LiveCGIToken.sol#80-82) is never used and should be removed
EnumerableSet.length(EnumerableSet.UintSet) (LiveCGIToken.sol#145-147) is never used and should be removed
EnumerableSet.remove(EnumerableSet.Bytes32Set,bytes32) (LiveCGIToken.sol#72-74) is never used and should be removed
EnumerableSet.remove(EnumerableSet.UintSet,uint256) (LiveCGIToken.sol#137-139) is never used and should be removed
EnumerableSet.values(EnumerableSet.AddressSet) (LiveCGIToken.sol#117-126) is never used and should be removed
EnumerableSet.values(EnumerableSet.Bytes32Set) (LiveCGIToken.sol#88-90) is never used and should be removed
EnumerableSet.values(EnumerableSet.UintSet) (LiveCGIToken.sol#153-162) is never used and should be removed
LiveCGIToken.resetTokenRoyalty(uint256) (LiveCGIToken.sol#1348-1354) is never used and should be removed
Strings.toHexString(uint256) (LiveCGIToken.sol#205-216) 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 (LiveCGIToken.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (LiveCGIToken.sol#237-242):
        - (success) = recipient.call{value: amount}() (LiveCGIToken.sol#240)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (LiveCGIToken.sol#264-275):
        - (success,returndata) = target.call{value: value}(data) (LiveCGIToken.sol#273)
Low level call in Address.functionStaticCall(address,bytes,string) (LiveCGIToken.sol#281-290):
        - (success,returndata) = target.staticcall(data) (LiveCGIToken.sol#288)
Low level call in Address.functionDelegateCall(address,bytes,string) (LiveCGIToken.sol#296-305):
        - (success,returndata) = target.delegatecall(data) (LiveCGIToken.sol#303)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter ERC2981.royaltyInfo(uint256,uint256)._tokenId (LiveCGIToken.sol#576) is not in mixedCase
Parameter ERC2981.royaltyInfo(uint256,uint256)._salePrice (LiveCGIToken.sol#576) is not in mixedCase
Parameter LiveCGIToken.addBlacklist(address)._user (LiveCGIToken.sol#1167) is not in mixedCase
Parameter LiveCGIToken.removeBlacklist(address)._user (LiveCGIToken.sol#1176) is not in mixedCase
Parameter LiveCGIToken.uri(uint256)._id (LiveCGIToken.sol#1202) is not in mixedCase
Parameter LiveCGIToken.setMessageProxy(address)._messageProxy (LiveCGIToken.sol#1221) is not in mixedCase
Parameter LiveCGIToken.setTargetChainHash(bytes32)._targetChainHash (LiveCGIToken.sol#1229) is not in mixedCase
Parameter LiveCGIToken.setTargetContract(address)._targetContract (LiveCGIToken.sol#1237) is not in mixedCase
```

```
Parameter LiveCGIToken.setTargetChainHash(bytes32)._targetChainHash (LiveCGIToken.sol#1229) is not in mixedCase
Parameter LiveCGIToken.setTargetContract(address)._targetContract (LiveCGIToken.sol#1237) is not in mixedCase
Parameter LiveCGIToken.setRoyaltyfeeNumerator(uint96)._royaltyfeeNumerator (LiveCGIToken.sol#1356) is not in mixedCase
Variable LiveCGIToken._baseUri (LiveCGIToken.sol#1159) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
grantRole(bytes32,address) should be declared external:
        - AccessControl.grantRole(bytes32,address) (LiveCGIToken.sol#474-476)
revokeRole(bytes32,address) should be declared external:
        - AccessControl.revokeRole(bytes32,address) (LiveCGIToken.sol#478-480)
renounceRole(bytes32,address) should be declared external:
        - AccessControl.renounceRole(bytes32,address) (LiveCGIToken.sol#482-486)
getRoleMember(bytes32,uint256) should be declared external:
        - AccessControlEnumerable.getRoleMember(bytes32,uint256) (LiveCGIToken.sol#522-524)
getRoleMemberCount(bytes32) should be declared external:
        - AccessControlEnumerable.getRoleMemberCount(bytes32) (LiveCGIToken.sol#526-528)
onERC1155Received(address,address,uint256,uint256,bytes) should be declared external:
        - ERC1155Holder.onERC1155Received(address,address,uint256,uint256,bytes) (LiveCGIToken.sol#542-550)
onERC1155BatchReceived(address,address,uint256[],uint256[],bytes) should be declared external:
        - ERC1155Holder.onERC1155BatchReceived(address,address,uint256[],uint256[],bytes) (LiveCGIToken.sol#552-560)
uri(uint256) should be declared external:
        - ERC1155.uri(uint256) (LiveCGIToken.sol#645-647)
        - LiveCGIToken.uri(uint256) (LiveCGIToken.sol#1202-1210)
balanceOfBatch(address[],uint256[]) should be declared external:
        - ERC1155.balanceOfBatch(address[],uint256[]) (LiveCGIToken.sol#654-670)
setApprovalForAll(address,bool) should be declared external:
        - ERC1155.setApprovalForAll(address,bool) (LiveCGIToken.sol#672-674)
safeTransferFrom(address,address,uint256,uint256,bytes) should be declared external:
        - ERC1155.safeTransferFrom(address,address,uint256,uint256,bytes) (LiveCGIToken.sol#680-692)
safeBatchTransferFrom(address,address,uint256[],uint256[],bytes) should be declared external:
        - ERC1155.safeBatchTransferFrom(address,address,uint256[],uint256[],bytes) (LiveCGIToken.sol#694-706)
burn(address,uint256,uint256) should be declared external:
        - ERC1155Burnable.burn(address,uint256,uint256) (LiveCGIToken.sol#926-937)
burnBatch(address,uint256[],uint256[]) should be declared external:
        - ERC1155Burnable.burnBatch(address,uint256[],uint256[]) (LiveCGIToken.sol#939-950)
mint(address,uint256,uint256,bytes) should be declared external:
        - ERC1155PresetMinterPauser.mint(address,uint256,uint256,bytes) (LiveCGIToken.sol#1089-1098)
```

```
mint(address,uint256,uint256,bytes) should be declared external:
        - ERC1155PresetMinterPauser.mint(address,uint256,uint256,bytes) (LiveCGIToken.sol#1089-1098)
mintBatch(address,uint256[],uint256[],bytes) should be declared external:
        - ERC1155PresetMinterPauser.mintBatch(address,uint256[],uint256[],bytes) (LiveCGIToken.sol#1100-1109)
pause() should be declared external:
        - ERC1155PresetMinterPauser.pause() (LiveCGIToken.sol#1111-1114)
unpause() should be declared external:
        - ERC1155PresetMinterPauser.unpause() (LiveCGIToken.sol#1116-1119)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:LiveCGIToken.sol analyzed (25 contracts with 75 detectors), 84 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

LiveCGIToken.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 264:4:

Low level calls:

Use of "delegatecall": should be avoided whenever possible. External code, that is called can change the state of the calling contract and send ether from the caller's balance. If this is wanted behaviour, use the Solidity library feature if possible.
more
Pos: 303:50:

Gas & Economy

Gas costs:

Gas requirement of function LiveCGIToken.postMessage 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: 1267:4:

Gas costs:

Gas requirement of function LiveCGIToken.supportsInterface 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: 1369:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 1331:8:

Miscellaneous

Constant/View/Pure functions:

MessageProxy.postOutgoingMessage(bytes32,address,bytes) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1144:4:

Constant/View/Pure functions:

LiveCGIToken._beforeTokenTransfer(address,address,address,uint256[],uint256[],bytes) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1384:4:

Similar variable names:

LiveCGIToken._mint(address,uint256,uint256,bytes) : Variables have very similar names "to" and "id". Note: Modifiers are currently not considered by this static analysis.
Pos: 1297:24:

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: 1395:8:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 621:8:

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: 589:32:

Solhint Linter

LiveCGIToken.sol

```
LiveCGIToken.sol:723:18: Error: Parse error: missing ';' at '{'
LiveCGIToken.sol:753:22: Error: Parse error: missing ';' at '{'
LiveCGIToken.sol:821:18: Error: Parse error: missing ';' at '{'
LiveCGIToken.sol:846:22: 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