Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer: Gull Token
Website: http://polygod.io
Platform: Binance Smart Chain
Language: Solidity
Date: October 26th, 2021

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 Gull team to perform the Security audit of the Gull 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 October 26th, 2021.

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 GULL token is a BEP20 standard token smart contract with other customization like: swapping. This audit only considers GULL token smart contract, and does not cover any other smart contracts in the platform.

Audit scope

Name	**Code Review and Security Analysis Report for Gull Token Smart Contract**
Platform	**BSC / Solidity**
File	GullToken.sol
File MD5 Hash	647E1BD3B943333E8433C216504C88DB
Updated File MD5 Hash	D4326137C1E26E80E7750952475534AA
Audit Date	October 26th, 2021
Revised Audit Date	October 28th, 2021

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Gull • Symbol: GULL • Decimals: 18 • Total Tokens: 13,200,000	**YES, This is valid.**
• Maximum Transaction Amount: 3 Million GULL • Minimum Tokens Before Swap: 1000 GULL • Capped Withdrawal Time Span: 1 Day • Capped Withdrawal Limit: 20000 GULL	**YES, This is valid.**
• Dev Fee: 6% • Community Fee: 2% • Liquidity Fee: 2%	**YES, This is valid.** **Owner authorized wallet can set some percentage value and we suggest handling the private key of that wallet securely.**

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 2 low and some very low level issues. These issues are not critical ones, so it's good to go for the production.

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	Moderated
	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	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**

Email: audit@EtherAuthority.io

Code Quality

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

The libraries in Gull 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 Gull Token.

The Gull 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 Gull Token smart contracts code in the form of a github 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://polygod.io/ 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	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	lockTheSwap	modifier	Passed	No Issue
3	receive	external	Passed	No Issue
4	mint	external	access only Owner	No Issue
5	burn	external	access only Owner	No Issue
6	addAdminRole	write	access only Owner/Admin Role	No Issue
7	revokeAdminRole	write	access only Owner/Admin Role	No Issue
8	adminRole	read	Passed	No Issue
9	updateCappedWithdrawal	external	access only Owner/Admin Role	No Issue
10	updateCappedWithdrawalTime	external	access only Owner/Admin Role	No Issue
11	excludeFromFees	write	access only Owner/Admin Role	No Issue
12	setAutomatedMarketMakerPair	write	access only Owner/Admin Role	No Issue
13	excludeFromCap	write	access only Owner/Admin Role	No Issue
14	isExcludedFromFees	read	Passed	No Issue
15	isExcludedFromCap	read	Passed	No Issue
16	_transferFeesWallets	write	Critical operation lacks event log	Refer Audit Findings
17	updateCappedWithdrawalToogle	external	access only Owner/Admin Role	No Issue
18	updateSwapTokenAmount	external	access only Owner/Admin Role	No Issue
19	updateSwapToogle	external	access only Owner/Admin Role	No Issue
20	updateFeesToogle	external	access only Owner/Admin Role	No Issue
21	updateFees	external	Critical operation lacks event log	Refer Audit Findings
22	updateUniswapV2Router	external	access only Owner/Admin Role	No Issue
23	_transfer	internal	Passed	No Issue
24	acceptWithdraw	internal	Passed	No Issue
25	userWithdrawAmount	read	Passed	No Issue
26	userWithdrawTime	read	Passed	No Issue
27	_partialFee	internal	Passed	No Issue
28	_calculateFeeAmount	internal	Critical operation lacks event log,	Refer Audit Findings

			Function input parameters lack of check	
29	swapTokensForEth	internal	Passed	No Issue
30	getTokenPrice	read	Passed	No Issue
31	withdrawTokenFunds	external	Critical operation lacks event log	Refer Audit Findings
32	name	read	Passed	No Issue
33	symbol	read	Passed	No Issue
34	decimals	read	Passed	No Issue
35	totalSupply	read	Passed	No Issue
36	balanceOf	read	Passed	No Issue
37	transfer	write	Passed	No Issue
38	allowance	read	Passed	No Issue
39	approve	write	Passed	No Issue
40	transferFrom	write	Passed	No Issue
41	increaseAllowance	write	Passed	No Issue
42	decreaseAllowance	write	Passed	No Issue
43	_transfer	internal	Passed	No Issue
44	_mint	internal	Passed	No Issue
45	_burn	internal	Passed	No Issue
46	_approve	internal	Passed	No Issue
47	_beforeTokenTransfer	internal	Passed	No Issue
48	_afterTokenTransfer	internal	Passed	No Issue
49	onlyRole	modifier	Passed	No Issue
50	supportsInterface	read	Passed	No Issue
51	hasRole	read	Passed	No Issue
52	_checkRole	internal	Passed	No Issue
53	getRoleAdmin	read	Passed	No Issue
54	grantRole	write	Passed	No Issue
55	revokeRole	write	Passed	No Issue
56	renounceRole	write	Passed	No Issue
57	_setupRole	internal	Passed	No Issue
58	_setRoleAdmin	internal	Passed	No Issue
59	_grantRole	internal	Passed	No Issue
60	_revokeRole	internal	Passed	No Issue
61	supportsInterface	read	Passed	No Issue
62	_msgSender	internal	Passed	No Issue
63	_msgData	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.

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) Critical operation lacks event log:

Missing event log for.

- withdrawTokenFunds
- _transferFeesWallets
- updateFees
- updateUniswapV2Router
- _calculateFeeAmount

Resolution: Please write an event log for listed events.

Status: Acknowledged

(2) Function input parameters lack of check:

Some functions require validation before execution.

Functions are:

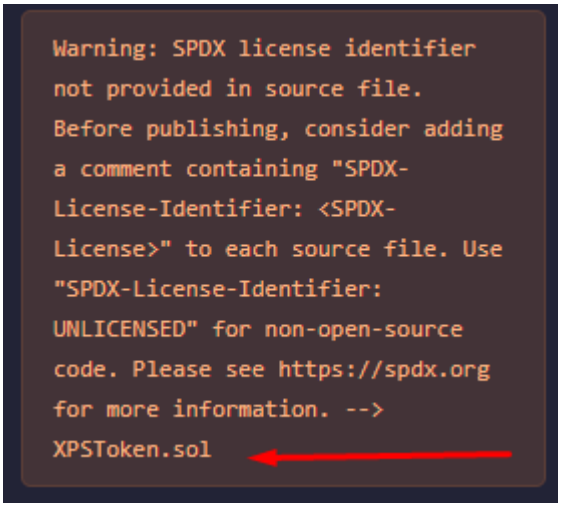
- _calculateFeeAmount
- withdrawTokenFunds

Resolution: Use validation : variable should be greater than 0 and for address type check variable is not address(0).

Status: Fixed

Very Low / Informational / Best practices:

(1) Warning: SPDX license identifier:

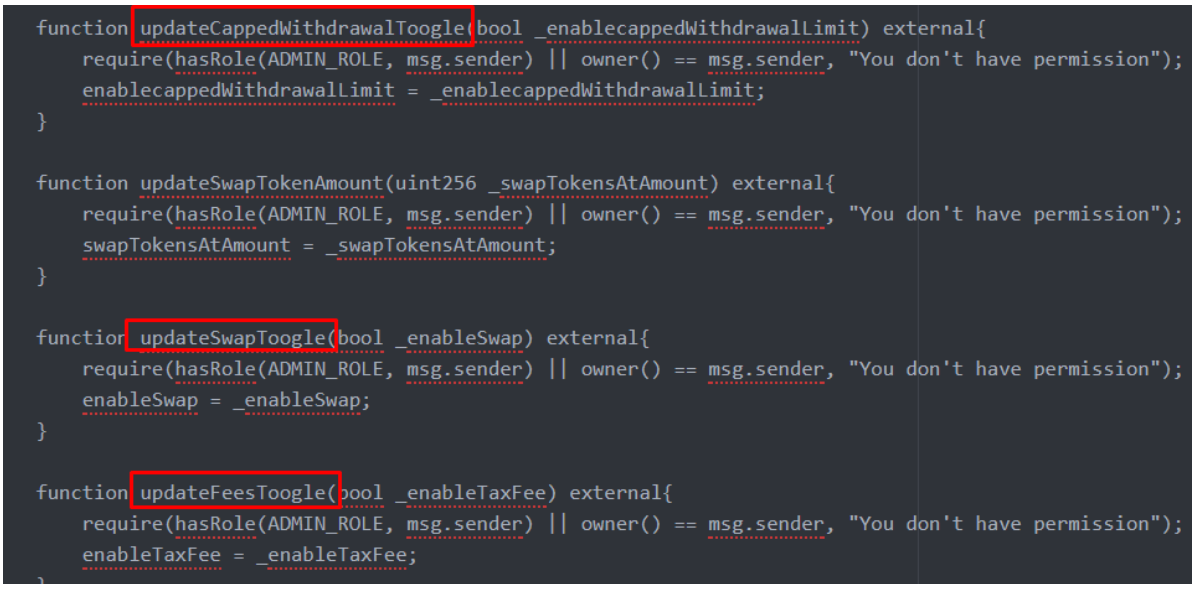


SPDX license identifier not provided in source file in IUniswapV2Factory.sol , IUniswapV2Router02.sol , IUniswapV2Pair.sol.

Resolution: SPDX-License-Identifier.

Status: Fixed

(2) Spelling mistake



```
function updateCappedWithdrawalToogle(bool _enablecappedWithdrawalLimit) external{
    require(hasRole(ADMIN_ROLE, msg.sender) || owner() == msg.sender, "You don't have permission");
    enablecappedWithdrawalLimit = _enablecappedWithdrawalLimit;
}

function updateSwapTokenAmount(uint256 _swapTokensAtAmount) external{
    require(hasRole(ADMIN_ROLE, msg.sender) || owner() == msg.sender, "You don't have permission");
    swapTokensAtAmount = _swapTokensAtAmount;
}

function updateSwapToogle(bool _enableSwap) external{
    require(hasRole(ADMIN_ROLE, msg.sender) || owner() == msg.sender, "You don't have permission");
    enableSwap = _enableSwap;
}

function updateFeesToogle(bool _enableTaxFee) external{
    require(hasRole(ADMIN_ROLE, msg.sender) || owner() == msg.sender, "You don't have permission");
    enableTaxFee = _enableTaxFee;
```

Spelling mistake in function name.

Functions are:

- updateCappedWithdrawalToogle
- updateSwapToogle

- updateFeesToogle

Toogle word should be Toggle.

Resolution: Correct the spelling.

Status: Fixed

Centralization

These smart contracts have some functions which can be executed by the Admin (Owner) or User having admin role only. Following are Admin functions:

- mint: Owner can check Exceeded the capped amount and mint amount.
- burn: Owner can burn the amount from any user's wallet address.
- addAdminRole: Owner/User having admin role can add admin role.
- revokeAdminRole: Owner/User having admin role can revoke admin role.
- withdrawTokenFunds: Owner/User having admin role can withdraw token funds from the wallet amount.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts, but they are not critical ones. Some of the issues have been resolved / acknowledged 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".**

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).

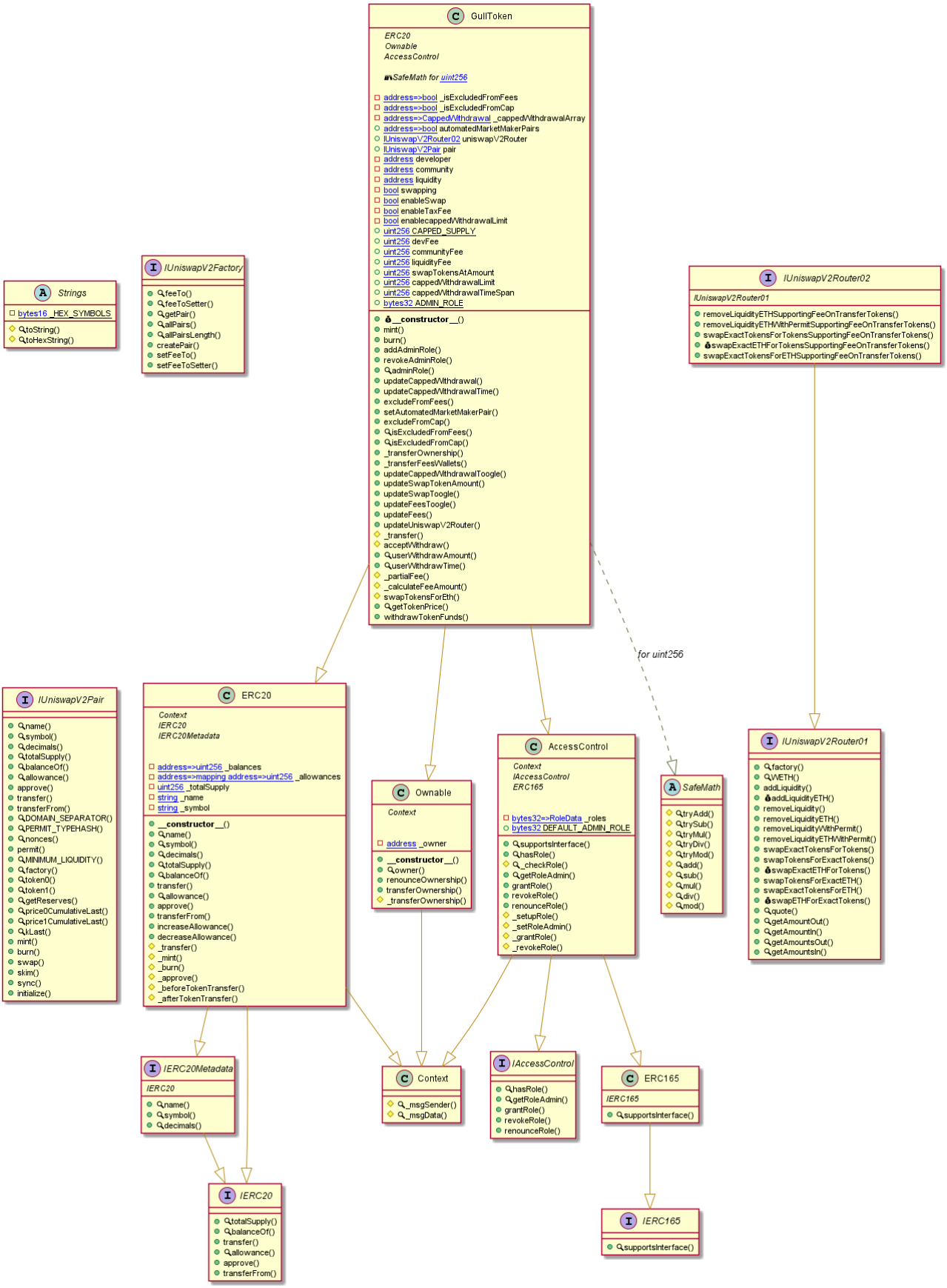
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Slither Results Log

Slither log >> GullToken.sol

```
INFO:Detectors:
GullToken.withdrawTokenFunds(address,address) (GullToken.sol#1499-1502) ignores return value by ercToken.transfer(wallet,ercToken.balance
Of(address(this))) (GullToken.sol#1501)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
Ownable._owner (GullToken.sol#266) is never initialized. It is used in:
        - Ownable.owner() (GullToken.sol#280-282)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
Contract locking ether found:
        Contract GullToken (GullToken.sol#1216-1505) has payable functions:
         - GullToken.receive() (GullToken.sol#1285-1287)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
Reentrancy in GullToken._calculateFeeAmount(address,uint256) (GullToken.sol#1444-1473):
        External calls:
        - swapTokensForEth(devFeeAmount,developer) (GullToken.sol#1455)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,receiver,block.timestamp + (60 *
1000)) (GullToken.sol#1483-1489)
        State variables written after the call(s):
        - super._transfer(address(this),community,communityFeeAmount) (GullToken.sol#1458)
                - _balances[sender] = senderBalance - amount (GullToken.sol#517)
                - _balances[recipient] += amount (GullToken.sol#519)
        - super._transfer(address(this),liquidity,liquidityFeeAmount) (GullToken.sol#1461)
                - _balances[sender] = senderBalance - amount (GullToken.sol#517)
                - _balances[recipient] += amount (GullToken.sol#519)
        - super._transfer(from,address(this),totalFeeAmount) (GullToken.sol#1470)
                - _balances[sender] = senderBalance - amount (GullToken.sol#517)
                - _balances[recipient] += amount (GullToken.sol#519)
Reentrancy in GullToken._transfer(address,address,uint256) (GullToken.sol#1396-1406):
        External calls:
        - newAmount = _partialFee(from,amount) (GullToken.sol#1402)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,receiver,block.timestamp + (60 *
1000)) (GullToken.sol#1483-1489)
        State variables written after the call(s):
        - super._transfer(from,to,newAmount) (GullToken.sol#1405)
```

```
                - _balances[sender] = senderBalance - amount (GullToken.sol#517)
                - _balances[recipient] += amount (GullToken.sol#519)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
GullToken.burn(address,uint256).owner (GullToken.sol#1294) shadows:
        - Ownable.owner() (GullToken.sol#280-282) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
GullToken.updateCappedWithdrawal(uint256) (GullToken.sol#1310-1314) should emit an event for:
        - cappedWithdrawalLimit = _cappedWithdrawalLimit (GullToken.sol#1313)
GullToken.updateCappedWithdrawalTime(uint256) (GullToken.sol#1316-1320) should emit an event for:
        - cappedWithdrawalTimeSpan = _cappedWithdrawalTimeSpan (GullToken.sol#1319)
GullToken.updateSwapTokenAmount(uint256) (GullToken.sol#1368-1371) should emit an event for:
        - swapTokensAtAmount = _swapTokensAtAmount (GullToken.sol#1370)
GullToken.updateFees(uint256,uint256,uint256) (GullToken.sol#1383-1388) should emit an event for:
        - devFee = _devFee (GullToken.sol#1385)
        - communityFee = _communityFee (GullToken.sol#1386)
        - liquidityFee = _liquidityFee (GullToken.sol#1387)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
GullToken._transferFeesWallets(address,address,address).newOwnerDev (GullToken.sol#1348) lacks a zero-check on :
                - developer = newOwnerDev (GullToken.sol#1350)
GullToken._transferFeesWallets(address,address,address).newOwnerCom (GullToken.sol#1348) lacks a zero-check on :
                - community = newOwnerCom (GullToken.sol#1351)
GullToken._transferFeesWallets(address,address,address).newOwnerLiq (GullToken.sol#1348) lacks a zero-check on :
                - liquidity = newOwnerLiq (GullToken.sol#1352)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in GullToken._calculateFeeAmount(address,uint256) (GullToken.sol#1444-1473):
        External calls:
        - swapTokensForEth(devFeeAmount,developer) (GullToken.sol#1455)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,receiver,block.timestamp + (60 *
1000)) (GullToken.sol#1483-1489)
        State variables written after the call(s):
        - swapping = false (GullToken.sol#1463)
Reentrancy in GullToken.constructor() (GullToken.sol#1252-1283):
        External calls:
        - _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (GullToken.sol
```

```
#1257-1258)
        State variables written after the call(s):
        - _mint(owner(),13200000 * (10 ** 18)) (GullToken.sol#1282)
                - _balances[account] += amount (GullToken.sol#541)
        - excludeFromCap(address(this),true) (GullToken.sol#1270)
                - _isExcludedFromCap[account] = excluded (GullToken.sol#1335)
        - excludeFromCap(address(_uniswapV2Router),true) (GullToken.sol#1271)
                - _isExcludedFromCap[account] = excluded (GullToken.sol#1335)
        - excludeFromCap(owner(),true) (GullToken.sol#1272)
                - _isExcludedFromCap[account] = excluded (GullToken.sol#1335)
        - excludeFromCap(_uniswapV2Pair,true) (GullToken.sol#1273)
                - _isExcludedFromCap[account] = excluded (GullToken.sol#1335)
        - excludeFromCap(developer,true) (GullToken.sol#1274)
                - _isExcludedFromCap[account] = excluded (GullToken.sol#1335)
        - excludeFromCap(community,true) (GullToken.sol#1275)
                - _isExcludedFromCap[account] = excluded (GullToken.sol#1335)
        - excludeFromCap(liquidity,true) (GullToken.sol#1276)
                - _isExcludedFromCap[account] = excluded (GullToken.sol#1335)
        - excludeFromFees(address(this),true) (GullToken.sol#1264)
                - _isExcludedFromFees[account] = excluded (GullToken.sol#1324)
        - excludeFromFees(owner(),true) (GullToken.sol#1265)
                - _isExcludedFromFees[account] = excluded (GullToken.sol#1324)
        - excludeFromFees(developer,true) (GullToken.sol#1266)
                - _isExcludedFromFees[account] = excluded (GullToken.sol#1324)
        - excludeFromFees(community,true) (GullToken.sol#1267)
                - _isExcludedFromFees[account] = excluded (GullToken.sol#1324)
        - excludeFromFees(liquidity,true) (GullToken.sol#1268)
                - _isExcludedFromFees[account] = excluded (GullToken.sol#1324)
```

```
        - _mint(owner(),13200000 * (10 ** 18)) (GullToken.sol#1282)
                - _totalSupply += amount (GullToken.sol#540)
        - setAutomatedMarketMakerPair(_uniswapV2Pair,true) (GullToken.sol#1279)
                - automatedMarketMakerPairs[account] = value (GullToken.sol#1329)
        - setAutomatedMarketMakerPair(address(uniswapV2Router),true) (GullToken.sol#1280)
                - automatedMarketMakerPairs[account] = value (GullToken.sol#1329)
        - pair = IUniswapV2Pair(_uniswapV2Pair) (GullToken.sol#1262)
        - uniswapV2Router = _uniswapV2Router (GullToken.sol#1261)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in GullToken._calculateFeeAmount(address,uint256) (GullToken.sol#1444-1473):
        External calls:
        - swapTokensForEth(devFeeAmount,developer) (GullToken.sol#1455)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,receiver,block.timestamp + (60 *
1000)) (GullToken.sol#1483-1489)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (GullToken.sol#521)
                - super._transfer(from,address(this),totalFeeAmount) (GullToken.sol#1470)
        - Transfer(sender,recipient,amount) (GullToken.sol#521)
                - super._transfer(address(this),liquidity,liquidityFeeAmount) (GullToken.sol#1461)
        - Transfer(sender,recipient,amount) (GullToken.sol#521)
                - super._transfer(address(this),community,communityFeeAmount) (GullToken.sol#1458)
Reentrancy in GullToken._transfer(address,address,uint256) (GullToken.sol#1396-1406):
        External calls:
        - newAmount = _partialFee(from,amount) (GullToken.sol#1402)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,receiver,block.timestamp + (60 *
1000)) (GullToken.sol#1483-1489)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (GullToken.sol#521)
                - super._transfer(from,to,newAmount) (GullToken.sol#1405)
Reentrancy in GullToken.constructor() (GullToken.sol#1252-1283):
        External calls:
        - _uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (GullToken.sol
#1257-1258)
        Event emitted after the call(s):
        - Transfer(address(0),account,amount) (GullToken.sol#542)
                - _mint(owner(),13200000 * (10 ** 18)) (GullToken.sol#1282)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
INFO:Detectors:
GullToken.acceptWithdraw(address,uint256) (GullToken.sol#1408-1430) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp.sub(_cappedWithdrawalArray[from].time) >= cappedWithdrawalTimeSpan (GullToken.sol#1413)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
AccessControl._setRoleAdmin(bytes32,bytes32) (GullToken.sol#787-791) is never used and should be removed
Context._msgData() (GullToken.sol#260-263) is never used and should be removed
GullToken._calculateFeeAmount(address,uint256) (GullToken.sol#1444-1473) is never used and should be removed
GullToken._partialFee(address,uint256) (GullToken.sol#1440-1442) is never used and should be removed
GullToken._transfer(address,address,uint256) (GullToken.sol#1396-1406) is never used and should be removed
GullToken.acceptWithdraw(address,uint256) (GullToken.sol#1408-1430) is never used and should be removed
GullToken.swapTokensForEth(uint256,address) (GullToken.sol#1476-1490) is never used and should be removed
SafeMath.add(uint256,uint256) (GullToken.sol#1085-1089) is never used and should be removed
SafeMath.div(uint256,uint256) (GullToken.sol#1135-1138) is never used and should be removed
SafeMath.div(uint256,uint256,string) (GullToken.sol#1190-1193) is never used and should be removed
SafeMath.mod(uint256,uint256) (GullToken.sol#1152-1155) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (GullToken.sol#1210-1213) is never used and should be removed
SafeMath.mul(uint256,uint256) (GullToken.sol#1116-1121) is never used and should be removed
SafeMath.sub(uint256,uint256) (GullToken.sol#1101-1104) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (GullToken.sol#1170-1173) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (GullToken.sol#1024-1028) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (GullToken.sol#1060-1063) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (GullToken.sol#1070-1073) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (GullToken.sol#1045-1053) is never used and should be removed
SafeMath.trySub(uint256,uint256) (GullToken.sol#1035-1038) is never used and should be removed
Strings.toHexString(uint256) (GullToken.sol#208-219) is never used and should be removed
Strings.toString(uint256) (GullToken.sol#183-203) 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 (GullToken.sol#2) 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:
Function IUniswapV2Router01.WETH() (GullToken.sol#819) is not in mixedCase
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (GullToken.sol#981) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (GullToken.sol#982) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (GullToken.sol#999) is not in mixedCase
```

```
Parameter GullToken.updateCappedWithdrawal(uint256)._cappedWithdrawalLimit (GullToken.sol#1310) is not in mixedCase
Parameter GullToken.updateCappedWithdrawalTime(uint256)._cappedWithdrawalTimeSpan (GullToken.sol#1316) is not in mixedCase
Function GullToken._transferFeesWallets(address,address,address) (GullToken.sol#1348-1361) is not in mixedCase
Parameter GullToken.updateCappedWithdrawalToogle(bool)._enablecappedWithdrawalLimit (GullToken.sol#1363) is not in mixedCase
Parameter GullToken.updateSwapTokenAmount(uint256)._swapTokensAtAmount (GullToken.sol#1368) is not in mixedCase
Parameter GullToken.updateSwapToogle(bool)._enableSwap (GullToken.sol#1373) is not in mixedCase
Parameter GullToken.updateFeesToogle(bool)._enableTaxFee (GullToken.sol#1378) is not in mixedCase
Parameter GullToken.updateFees(uint256,uint256,uint256)._devFee (GullToken.sol#1383) is not in mixedCase
Parameter GullToken.updateFees(uint256,uint256,uint256)._communityFee (GullToken.sol#1383) is not in mixedCase
Parameter GullToken.updateFees(uint256,uint256,uint256)._liquidityFee (GullToken.sol#1383) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (GullToken.sol#261)" inContext (GullToken.sol#255-264)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (GullToken.sol#8
24) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (Gu
llToken.sol#825)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
GullToken.constructor() (GullToken.sol#1252-1283) uses literals with too many digits:
        - _mint(owner(),13200000 * (10 ** 18)) (GullToken.sol#1282)
GullToken.slitherConstructorConstantVariables() (GullToken.sol#1216-1505) uses literals with too many digits:
        - CAPPED_SUPPLY = 150000000 * (10 ** 18) (GullToken.sol#1239)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
Ownable._owner (GullToken.sol#266) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

```
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (GullToken.sol#299-301)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (GullToken.sol#307-310)
name() should be declared external:
        - ERC20.name() (GullToken.sol#345-347)
symbol() should be declared external:
        - ERC20.symbol() (GullToken.sol#353-355)
decimals() should be declared external:
        - ERC20.decimals() (GullToken.sol#370-372)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (GullToken.sol#396-399)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (GullToken.sol#404-406)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (GullToken.sol#415-418)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (GullToken.sol#433-447)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (GullToken.sol#461-464)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (GullToken.sol#480-488)
grantRole(bytes32,address) should be declared external:
        - AccessControl.grantRole(bytes32,address) (GullToken.sol#723-725)
renounceRole(bytes32,address) should be declared external:
        - AccessControl.renounceRole(bytes32,address) (GullToken.sol#754-758)
addAdminRole(address) should be declared external:
        - GullToken.addAdminRole(address) (GullToken.sol#1298-1300)
revokeAdminRole(address) should be declared external:
        - GullToken.revokeAdminRole(address) (GullToken.sol#1302-1304)
adminRole(address) should be declared external:
        - GullToken.adminRole(address) (GullToken.sol#1306-1308)
isExcludedFromFees(address) should be declared external:
        - GullToken.isExcludedFromFees(address) (GullToken.sol#1338-1340)
isExcludedFromCap(address) should be declared external:
        - GullToken.isExcludedFromCap(address) (GullToken.sol#1342-1344)
_transferFeesWallets(address,address,address) should be declared external:
```

```
_transferFeesWallets(address,address,address) should be declared external:
        - GullToken._transferFeesWallets(address,address,address) (GullToken.sol#1348-1361)
userWithdrawAmount(address) should be declared external:
        - GullToken.userWithdrawAmount(address) (GullToken.sol#1432-1434)
userWithdrawTime(address) should be declared external:
        - GullToken.userWithdrawTime(address) (GullToken.sol#1436-1438)
getTokenPrice(uint256) should be declared external:
        - GullToken.getTokenPrice(uint256) (GullToken.sol#1493-1497)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:GullToken.sol analyzed (16 contracts with 75 detectors), 84 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
root@server:/chetan/gaza/mycontracts#
```

Solidity Static Analysis

GullToken.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in GullToken.(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1252:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in GullToken.swapTokensForEth(uint256,address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1476:5:

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: 1413:19:

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: 1421:60:

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: 1488:12:

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: 345:4:

Email: audit@EtherAuthority.io

Gas costs:

Gas requirement of function GullToken.uniswapV2Router 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: 1228:4:

Gas costs:

Gas requirement of function GullToken.pair 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: 1229:4:

Gas costs:

Gas requirement of function GullToken.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: 1289:4:

Gas costs:

Gas requirement of function GullToken.getTokenPrice 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: 1493:3:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 972:4:

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: 23:4:

Constant/View/Pure functions:

IERC20.approve(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 48:4:

Similar variable names:

GullToken.swapTokensForEth(uint256,address) : Variables have very similar names "pair" and "path". Note: Modifiers are currently not considered by this static analysis.
Pos: 1478:8:

Similar variable names:

GullToken.swapTokensForEth(uint256,address) : Variables have very similar names "pair" and "path". Note: Modifiers are currently not considered by this static analysis.
Pos: 1479:8:

Similar variable names:

GullToken.swapTokensForEth(uint256,address) : Variables have very similar names "pair" and "path". Note: Modifiers are currently not considered by this static analysis.
Pos: 1480:8:

Similar variable names:

GullToken.swapTokensForEth(uint256,address) : Variables have very similar names "pair" and "path". Note: Modifiers are currently not considered by this static analysis.
Pos: 1486:12:

No return:

IERC20.totalSupply(): Defines a return type but never explicitly returns a value.
Pos: 9:4:

No return:

IERC20.balanceOf(address): Defines a return type but never explicitly returns a value.
Pos: 14:4:

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: 1290:8:

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: 1311:8:

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: 1312:8:

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: 1317:8:

Solhint Linter

GullToken.sol

```
GullToken.sol:442:18: Error: Parse error: missing ';' at '{'
GullToken.sol:483:18: Error: Parse error: missing ';' at '{'
GullToken.sol:516:18: Error: Parse error: missing ';' at '{'
GullToken.sol:565:18: 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