Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Jesus Coin
Platform:	Ethereum
Language:	Solidity
Date:	January 27th, 2024

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

As part of EtherAuthority's community smart contracts audit initiatives, the Jesus Coin smart contract from JESUS was audited extensively. 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 January 27th, 2024.

The purpose of this audit was to address the following:

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

Project Background

- The Jesus Coin is an ERC20-based smart contract in which the owner can update taxes, update charity wallet addresses, include or exclude any user from fees and burn his own tokens.
- This contract demonstrates a comprehensive implementation of a token with built-in transaction fees directed to a charity wallet, integrating with Uniswap for liquidity, and providing standard ERC20 functionalities with additional ownership controls.

Audit scope

Name	**Code Review and Security Analysis Report for Jesus Coin Smart Contract**
Platform	**Ethereum**
Language	**Solidity**
File	JESUS.sol
Ethereum Code	0xa8074c8d43e0a50fe8b1ae709b2fe77b9591b046
Audit Date	January 27th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Jesus Coin • Symbol: JESUS • Decimals: 18 • 0.7% fee on buy and sell. • Total Supply: 700 Trillion • 0.7% of the total supply will be transferred to the team wallet.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**
Ownership control: • The owner can update the address status from whiteList. • The owner can update buy tax and sell tax fees to a maximum of 25%. • The owner can update the charity wallet address. • The owner can burn his tokens. • The current owner can transfer the ownership. • The owner can renounce ownership.	**YES, This is valid. We suggest renouncing ownership to make the contract fully decentralized.**

Audit Summary

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

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

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

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

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

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack 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	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Business Risk Analysis

Category	Result
Buy Tax	0.7%
Sell Tax	0.7%
Cannot Buy	Yes
Cannot Sell	Yes
Max Tax	25%
Modify Tax	Yes
Fee Check	Yes
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	No
Pause Transfer?	No
Max Tax?	No
Is it Anti-whale?	No
Is Anti-bot?	Not Detected
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	No
Is it Proxy?	No
Can Take Ownership?	Yes
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

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

The libraries in Jesus Coin 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 Jesus Coin.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are not well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Jesus Coin smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are not well commented on. but the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

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

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

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Hardcoded values	Refer Audit Findings
2	_transfer	internal	Passed	No Issue
3	whiteListFromFee	write	Function input parameters lack of check, Centralization Risks	No Issue
4	includeInFee	write	Function input parameters lack of check, Centralization Risks	No Issue
5	changeTaxes	write	Critical operation lacks an event log and Centralization Risks	No Issue
6	changeCharityWallet	write	Critical operation lacks event log, Function input parameters lack check, Centralization Risks	No Issue
7	burnTokens	write	Centralization Risks	Refer Audit Findings
8	name	read	Passed	No Issue
9	symbol	read	Passed	No Issue
10	decimals	read	Passed	No Issue
11	totalSupply	read	Passed	No Issue
12	balanceOf	read	Passed	No Issue
13	transfer	write	Passed	No Issue
14	allowance	read	Passed	No Issue
15	approve	write	Passed	No Issue
16	transferFrom	write	Passed	No Issue
17	increaseAllowance	write	Passed	No Issue
18	decreaseAllowance	write	Passed	No Issue
19	_transfer	internal	Passed	No Issue
20	_mint	internal	Passed	No Issue
21	_burn	internal	Passed	No Issue
22	_approve	internal	Passed	No Issue
23	_spendAllowance	internal	Passed	No Issue
24	_beforeTokenTransfer	internal	Passed	No Issue
25	_afterTokenTransfer	internal	Passed	No Issue
26	onlyOwner	modifier	Passed	No Issue
27	owner	read	Passed	No Issue
28	_checkOwner	internal	Passed	No Issue

29	renounceOwnership	write	Centralization Risks	Refer Audit Findings
30	transferOwnership	write	Centralization Risks	Refer Audit Findings
31	_transferOwnership	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 a 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 a 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:

- changeTaxes
- changeCharityWallet

Resolution: We suggest writing an event log for listed events.

(2) Function input parameters lack of check:

Variable validation is not performed in the below functions:

- changeCharityWallet = account
- whiteListFromFee = account
- includeInFee = account

Resolution: We suggest putting validation for the inputs like integer type variables should be greater than 0 and address type variables should not be addressed (0).

Very Low / Informational / Best practices:

(1) Hardcoded values:

```
contract JESUS is ERC20, Ownable {
    using SafeMath for uint256;

    uint256 public buyFee = 7; //0.7% percent fee on buy and sell
    uint256 public sellFee = 7;
    address public charityWallet = 0x353bd4704b8dE7B45d12E7dd33cD31725f120D7a;
    mapping (address => bool) public _isWhiteListedFromFee;


    IUniswapV2Router02 public immutable uniswapV2Router;
    address public immutable uniswapV2Pair;

    constructor() ERC20("Jesus Coin", "JESUS") {    ⛽ infinite gas 1843200 gas
        IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(
            0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
        );
        // CREATE A UNISWAP PAIR FOR THIS NEW TOKEN
        uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
```

The _uniswapV2Router variable value is hardcoded.

Resolution: We suggest always ensuring the values prior to contract deployment.

(2) Centralization Risks:

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

- transferOwnership
- whiteListFromFee
- includeInFee
- changeTaxes
- changeCharityWallet
- burnTokens
- renounceOwnership

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

```
function whiteListFromFee(address account) public onlyOwner {    27209 gas
    _isWhiteListedFromFee[account] = true;
}
```

The Owner can add the user to the whitelist. They can sell and buy without any fees.

Resolution: We strongly recommend making your smart contract fully decentralized.

Centralization

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

JESUS.sol

- whiteListFromFee: The owner can update the address status to true from whiteList.
- includeInFee: The owner can update the address status to false from whiteList.
- changeTaxes: The owner can update the buy tax and sell tax values.
- changeCharityWallet: The owner can update the charity wallet address.
- burnTokens: The owner can burn his tokens.

Ownable.sol

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

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

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We had observed 2 low and 2 Informational issues in the smart contracts. but those are not critical. So, **it's good to go for the production**.

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

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

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

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

Our Methodology

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

Manual Code Review:

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

Vulnerability Analysis:

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

Documenting Results:

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

Suggested Solutions:

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

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

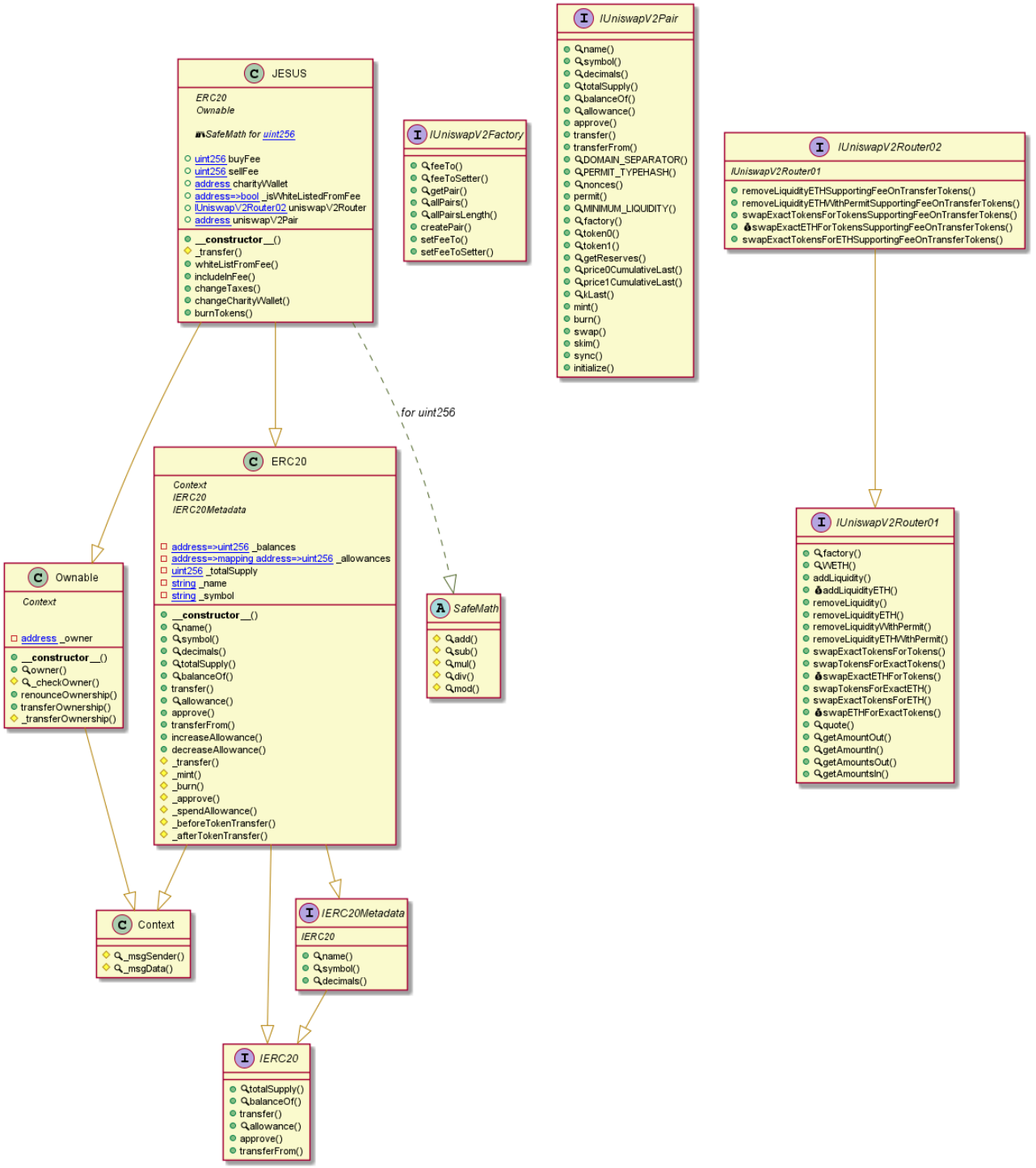
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

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

Appendix

Code Flow Diagram - Jesus Coin



Slither Results Log

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

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

Slither Log >> JESUS.sol

```
JESUS.changeTaxes(uint256,uint256) (JESUS.sol#1183-1188) should emit an event for:
        - buyFee = _buyTax (JESUS.sol#1186)
        - sellFee = _sellTax (JESUS.sol#1187)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

JESUS.changeCharityWallet(address).account (JESUS.sol#1190) lacks a zero-check on :
                - charityWallet = account (JESUS.sol#1191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Context._msgData() (JESUS.sol#25-27) is never used and should be removed
SafeMath.add(uint256,uint256) (JESUS.sol#657-662) is never used and should be removed
SafeMath.mod(uint256,uint256) (JESUS.sol#767-769) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (JESUS.sol#783-786) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.9 (JESUS.sol#6) allows old versions
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Function IUniswapV2Pair.DOMAIN_SEPARATOR() (JESUS.sol#849) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (JESUS.sol#851) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (JESUS.sol#882) is not in mixedCase
Function IUniswapV2Router01.WETH() (JESUS.sol#922) is not in mixedCase
Parameter JESUS.changeTaxes(uint256,uint256)._buyTax (JESUS.sol#1183) is not in mixedCase
Parameter JESUS.changeTaxes(uint256,uint256)._sellTax (JESUS.sol#1183) is not in mixedCase
Variable JESUS._isWhiteListedFromFee (JESUS.sol#1121) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (JESUS.
sol#927) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amou
ntBDesired (JESUS.sol#928)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

JESUS.constructor() (JESUS.sol#1127-1143) uses literals with too many digits:
        - _mint(msg.sender,700000000000000 * 10 ** decimals()) (JESUS.sol#1138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
JESUS.sol analyzed (11 contracts with 84 detectors), 17 result(s) found
```

Solidity Static Analysis

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

JESUS.sol

Check-effects-interaction:

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

more

Pos: 1127:4:

Gas costs:

Gas requirement of function JESUS.changeTaxes 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: 1183:4:

Gas costs:

Gas requirement of function JESUS.burnTokens 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: 1194:4:

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type

more

Pos: 828:4:

Email: audit@EtherAuthority.io

Constant/View/Pure functions:

JESUS._transfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1145:4:

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: 547:49:

No return:

IUniswapV2Router02.removeLiquidityETHWithPermitSupportingFeeOnTransf
Defines a return type but never explicitly returns a value.
Pos: 1078:4:

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: 1185:8:

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: 714:16:

Solhint Linter

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

JESUS.sol

```
Compiler version ^0.8.9 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:5
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:56
Error message for require is too long
Pos: 9:98
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:277
Error message for require is too long
Pos: 9:444
Error message for require is too long
Pos: 9:568
Code contains empty blocks
Pos: 24:617
Code contains empty blocks
Pos: 24:637
Error message for require is too long
Pos: 9:713
Function name must be in mixedCase
Pos: 5:848
Function name must be in mixedCase
Pos: 5:850
Function name must be in mixedCase
Pos: 5:881
Function name must be in mixedCase
Pos: 5:921
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:1126
```

Software analysis result:

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

Email: audit@EtherAuthority.io