Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	ARULSINGAM Token
Website:	arulsingam-arsi.com
Platform:	Binance Smart Chain
Language:	Solidity
Date:	March 21st, 2023

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 ARULSINGAM INVESTMENTS team to perform the Security audit of the ARSI 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 March 21st, 2023.

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 ARSI token supports a self-sufficient and self-reinforcing ecosystem that is directly linked with their companies' real estate assets.
- The ARSI token will facilitate all transactions within the ecosystem. Their real estate portfolio will constantly generate rental income in FIAT currencies that will be exchanged into ARSI by the team, therefore creating a steady demand for ARSI.
- ARSI is a BSC token that offers a multitude of various use cases for the real estate sector and its community.

Audit scope

Name	**Code Review and Security Analysis Report for ARSI Token Smart Contract**
Platform	**BSC / Solidity**
File	ARULSINGAM.sol
File MD5 Hash	C3C92CDE5CA5C78B24D09D4573F50A11
Revised Code MD5 Hash	A4306564FE38FA093EA67A14A5A641D6
Smart Contract Code	0x18B07afB2e337A0F24107f8a8b253a831f3EBcbE
Audit Date	March 21st, 2023
Revision Date	April 11th, 2023

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: ARULSINGAM • Symbol: ARSI • Decimals: 18 • Maximum Supply: 1 Billion • Maximum amount for a transaction: 2% of the maximum supply. • Maximum tokens limit of a wallet: 4% of the maximum supply. **Buy Fees:** • Developer Fees: 2% • Marketing Fees: 1% • Charity Fees: 1% • Burn Fees: 1% **Sell Fees:** • Developer Fees: 2% • Marketing Fees: 1% • Charity Fees: 1% • Burn Fees: 1%	**YES, This is valid.**
Ownership Control: • Trading status can be enabled by the owner. • Automated Market Maker Pair address can be set by the owner. • Owner can exclude the specified account from tax/Limits.	**YES, This is valid.**

Authorized Owner Control: ● Authorized Owner can set the tax for buy fees ● Authorized Owner can set the tax for sales fees.	

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, 1 high, 1 medium and 5 low and some very low level issues. And We confirm that these issues are fixed / acknowledged in the revised smart contract 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.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

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 ARSI 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 ARSI Token.

The ARSI 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 **well** commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given an ARULSINGAM Token smart contract code in the form of a bscscan web link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **well** commented. 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.

Another source of information was its official website: https://arulsingam-arsi.com which provided rich information about the project architecture and tokenomics.

Use of Dependencies

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

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_checkOwner	internal	Passed	removed
7	_transferOwnership	internal	Passed	removed
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
21	_transfer	internal	Passed	No Issue
22	_mint	internal	Passed	No Issue
23	_burn	internal	Passed	No Issue
24	_approve	internal	Passed	No Issue
25	_beforeTokenTransfer	internal	Passed	No Issue
26	_afterTokenTransfer	internal	Passed	removed
27	approveMax	write	Passed	No Issue
28	handleTax	write	Passed	removed
29	_transfer	internal	Passed	removed
30	enabledTrading	external	access only Owner	No Issue
31	setIsTimelockExempt	external	Passed	removed
32	setAutomatedMarketMakerPair	external	Passed	No Issue
33	cooldownEnabled	write	access only Owner	removed
34	addToWhitelist	write	access only Owner	removed
35	removeFromWhitelist	write	access only Owner	removed
36	setTxLimit	write	Passed	removed
37	setMaxTxPercent_base1000	write	Passed	removed
38	setMaxWalletPercent_base1000	write	Passed	removed
39	setIsTxLimitExempt	write	Passed	removed
40	excludeFromMaxWalletLimit	write	Passed	removed

41	includeInMaxWalletLimit	write	Passed	removed
42	swapBackSettings	write	access only Authorized	removed
43	setBuyFees	write	access only Authorized	No Issue
44	setSellFees	write	access only Authorized	No Issue
45	setFeeReceivers	write	access only Authorized	removed
46	isWhitelisted	read	Passed	removed
47	buyFees	read	Passed	removed
48	sellFees	read	Passed	removed
49	feeWallets	read	Passed	removed
50	multiTransfer	external	Passed	No Issue
51	multiTransfer_fixed	external	Passed	No Issue
52	clearStuckBalance	write	Owner can drain token balance of the contract	Refer to audit findings
53	clearBNB	write	Owner can drain all the BNB balance of contract	Refer to audit findings
54	forwardEther	internal	Passed	removed
55	authorize	external	Passed	No Issue
56	unauthorize	external	Passed	No Issue
57	onlyAuthorized	modifier	Passed	No Issue
58	receive	external	Passed	No Issue
59	enableTrading	external	The tradingEnabled once enabled then no option to disable it	Fixed
60	removeLimits	external	A limitsInEffect once disabled then no option to enable it	Fixed
61	excludeFromMaxTransaction	write	access only Owner	No Issue
62	excludeFromFees	write	access only Owner	No Issue
63	_setAutomatedMarketMakerPair	write	Passed	No Issue
64	isExcludedFromFees	read	Passed	No Issue
65	setFeeWallets	external	access only Owner	No Issue
66	transferFeesToWallet	write	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

(1) Logical vulnerability:

```
/**
 * @dev owner can remove any existing admin
 */

function unauthorize (address admin) external onlyOwner {
    require (!authorized[admin], "it's already unauthorized");
    authorized[admin] = false;
}
```

The required condition is wrong in an unauthorized function. because of this I am not able to unauthorize the address from the contract.

Resolution: We suggest correcting the condition in the required statement as below.

Status: This is fixed in the revised smart contract code.

.

Medium

(1) Perform multiplication on division:

```
*/
function handleTax(address from, address to, uint256 amount) private returns (uint256) {
    address[] memory sellPath = new address[](2);
    sellPath[0] = address(this);
    sellPath[1] = uniswapV2Router02.WETH();

    if(!isWhitelisted(from) && !isWhitelisted(to)) {
        if(!isExcludedFromMaxTx[from]){
        require(amount <= maxTxAmount, "Max Tx Amount exceeds");
        }
        uint256 tax;
        uint256 baseUnit = amount / denominator;
        if(automatedMarketMakerPairs[from] &&  buyCooldownEnabled &&
        !isTimelockExempt[to]) {
            require(cooldownTimer[to] < block.timestamp,"Please wait for cooldown between two buys");
            tax += baseUnit * buyTaxes["marketing"];
            tax += baseUnit * buyTaxes["dev"];
            tax += baseUnit * buyTaxes["charity"];
            tax += baseUnit * buyTaxes["burn"];
            if(tax > 0) {
                _transfer(from, address(this), tax);
            }
            cooldownTimer[to] = block.timestamp + cooldownTimerInterval;
            marketingTokens += baseUnit * buyTaxes["marketing"];
            devTokens += baseUnit * buyTaxes["dev"];

        } else if(automatedMarketMakerPairs[to]) {
            tax += baseUnit * sellTaxes["marketing"];
            tax += baseUnit * sellTaxes["dev"];
            tax += baseUnit * sellTaxes["charity"];
            tax += baseUnit * sellTaxes["burn"];


            if(tax > 0) {
                _transfer(from, address(this), tax);
            }

            marketingTokens += baseUnit * sellTaxes["marketing"];
            devTokens += baseUnit * sellTaxes["dev"];
            charityTokens += baseUnit * sellTaxes["charity"];
            burnTokens += baseUnit * sellTaxes["burn"];
```

The handleTax function performs a multiplication on the result of a division.

Resolution: we suggest performing multiplication first, and then performing division.

Status: This is a function removed in the revised smart contract code.

Low

(1) Owner can withdraw all the BNB balance of contract:

```
function clearBNB (address wallet) public onlyOwner {
    require(wallet!= address(0), "error: zero address");
    uint256 balance = address(this).balance;
    (bool s,) = wallet.call{value: balance}("");
   require(s);
}
```

Owner can drain all the BNB balance of the contract. If the private key of the owner's wallet is compromised, then it will create a problem.

Resolution: The owner can accept this risk and handle the private key very securely.

Status: This issue is acknowledged as a necessary function to run the business.

(2) Owner can withdraw token balance of the contract:

```
function clearStuckBalance (address token, address wallet) public onlyOwner {
    require (wallet != address(0), "error: zero address");
    uint256 balance =  IERC20(token).balanceOf(address(this));
    if (balance > 0){
     IERC20 (token).transfer(wallet, balance);
     }

}
```

Owner can drain all the token balance of the contract. If the private key of the owner's wallet is compromised, then it will create a problem.

Resolution: The owner can accept this risk and handle the private key very securely.

Status: This issue is acknowledged as a necessary function to run the business.

(3) Function input parameters lack of check:

Some functions require validation before execution.
Functions are:

- authorize
- clearBNB
- clearStuckBalance
- multiTransfer_fixed
- multiTransfer
- setAutomatedMarketMakerPair
- approveMax

Resolution: We suggest using validation like for numerical variables that should be greater than 0 and for address type check variables that are not address(0). For percentage type variables, values should have some range like minimum 0 and maximum 100.

Status: This is fixed in the revised smart contract code.

(4) ERC20.sol is not correct, missing some functions:

ERC20.sol is not correct, missing some functions.not matching with OpenZeppelin
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.8.2/contracts/token/ERC20/ERC20.sol
1. missing _afterTokenTransfer method
2. using safemath

Resolution: Suggest to use
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.8.2/contracts/token/ERC20/ERC20.sol
to eliminate the security risk.

Status: This is fixed in the revised smart contract code.

(5) Ownable.sol is not correct:

Ownable.sol is not correct,not matching with OpenZeppelin
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.8.2/contracts/access/Ownable.sol

Resolution: Suggest to use
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.8.2/contracts/access/Ownable.sol
to eliminate the security risk.

Status: This is fixed in the revised smart contract code.

Very Low / Informational / Best practices:

(1) SafeMath Library:

SafeMath Library is used in this contract code, but the compiler version is greater than or equal to 0.8.0, Then it will be not required to use, solidity automatically handles overflow/underflow.

Resolution: Remove the SafeMath library and use normal math operators, It will improve code size, and less gas consumption.

Status: This is fixed in the revised smart contract code.

(2) Unused variables:

```
uint256 private devTaxBuy;
uint256 private marketingTaxBuy;
uint256 private charityTaxBuy;
uint256 private burnTaxBuy;

uint256 private devTaxSell;
uint256 private marketingTaxSell;
uint256 private charityTaxSell;
uint256 private burnTaxSell;
```

```
address private devTaxWallet;
address private marketingTaxWallet;
address private charityTaxWallet;
address private burnTaxWallet;
```

There are variables defined but not used anywhere.

- devTaxBuy
- marketingTaxBuy
- charityTaxBuy
- burnTaxBuy
- devTaxSell
- marketingTaxSell
- charityTaxSell
- burnTaxSell
- devTaxWallet
- marketingTaxWallet
- charityTaxWallet
- burnTaxWallet

Resolution: Remove unused variables from the code.

Status: These variables are removed in the revised smart contract code.

(3) Variable should be immutable:

```
uint256 public maxTransactionAmount;
uint256 public swapTokensAtAmount;
uint256 public maxWallet;
```

Variables that are defined within the constructor but further remain unchanged should be marked as immutable to save gas and to ease the reviewing process of third-parties.

Resolution: We suggest declaring those variables as immutable.

Status: Fixed: Variables are declared as immutable.

(4) The unused SwapAndLiquify event:

```
event SwapAndLiquify(
    uint256 tokensSwapped,
    uint256 ethReceived,
    uint256 tokensIntoLiquidity
);
```

The SwapAndLiquify event is declared but not used.

Resolution: Suggest removing unused events.

Status: This is fixed in the revised smart contract code.

(5) A limitsInEffect once disabled then no option to enable it:

```
/**
 * @dev remove limits from contract
 */
function removeLimits() external onlyOwner returns (bool){
    limitsInEffect = false;
    return true;
}
```

A limitsInEffect once disabled then no option to enable it.

Resolution: Suggest to include an option to enable if necessary.

Status: This is fixed in the revised smart contract code.

(6) The tradingEnabled once enabled then no option to disable it:

```
/**
 * @dev enable trading, once enabled can't be turned off
 */
function enableTrading() external onlyOwner {
    tradingEnabled = true;
}
```

The tradingEnabled once enabled then no option to disable it.

Resolution: Suggest to include an option to disable if necessary.

Status: This is fixed in the revised smart contract code.

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:

ARULSINGAM.sol

- enabledTrading: Trading status can be enabledTrading by the owner.
- setAutomatedMarketMakerPair: Automated Market Maker Pair address can be set by the owner.
- setBuyFees: Authorized Owner can set the tax for buy fees.
- setSellFees: Authorized Owner can set the tax for sell fees..
- multiTransfer: The Owner can airdrop with different amounts.
- multiTransfer_fixed: The Owner can airdrop with different amounts.
- clearStuckBalance: The Owner can claim a stuck token from the token contract.
- clearBNB: The Owner can claim stuck BNB from the token contract.
- authorize: The Owner can authorize any address.
- unauthorize:The Owner can unauthorize any existing authorized address.
- enableTrading: Enable trading status can be set by the owner.
- removeLimits: Remove limits from contract by the owner.
- excludeFromMaxTransaction: An Exclude / include particular address from maximum transaction amount by the owner.
- excludeFromFees: An exclude/include particular address from fees by the owner.
- setFeeWallets: Sets fees wallets by the owner.

Ownable.sol

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

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 a Bscscan web link. And we have used all possible tests based on given objects as files. We had observed 1 high severity issue, 5 low severity issues and some Informational issues in the smart contracts. We confirm that these issues are fixed / acknowledged in the revised smart contract code. So, **it's good to go for the mainnet deployment**.

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".**

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).

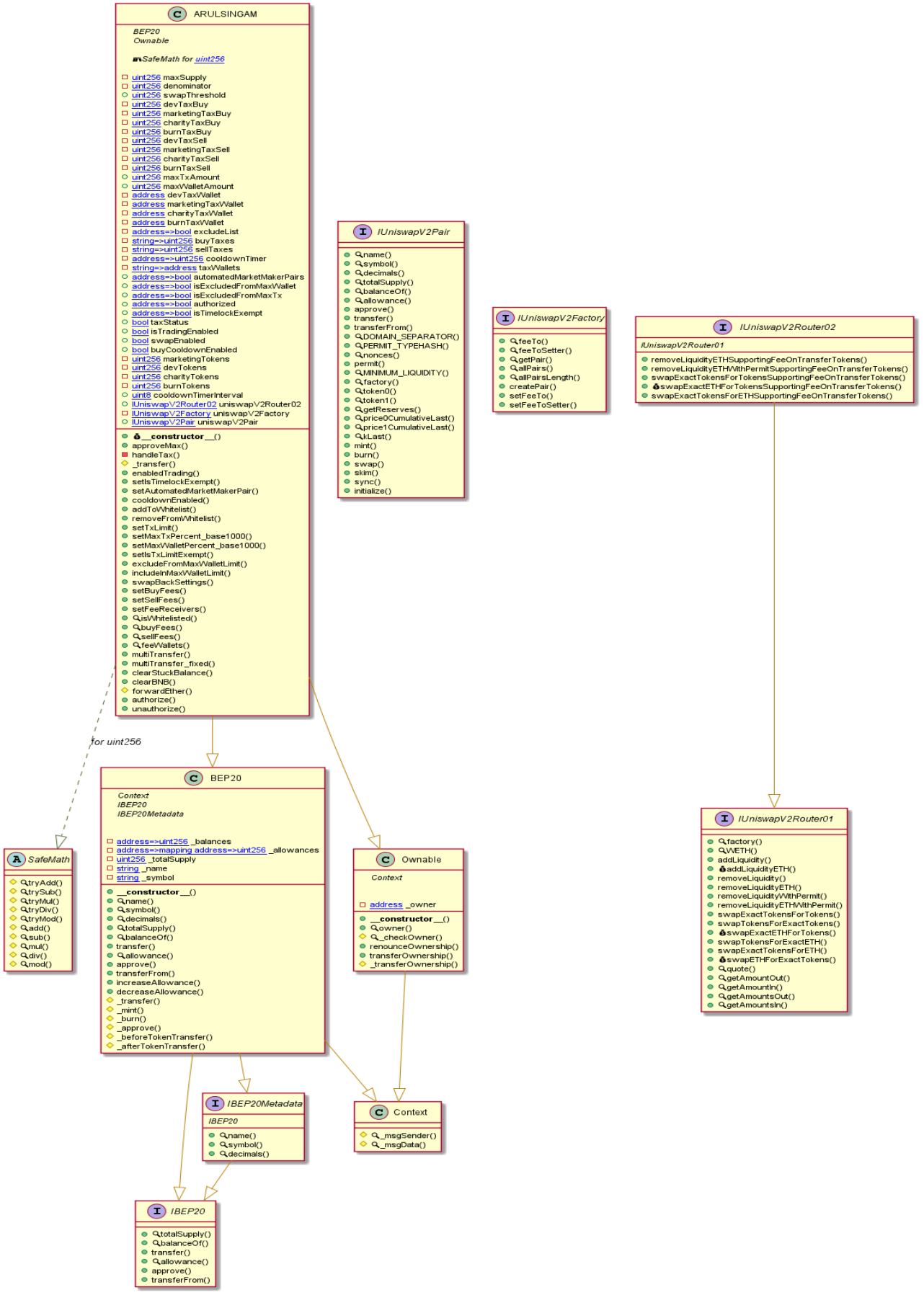
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Slither Results Log

Slither Log >> ARULSINGAM.sol

```
ARULSINGAM.cooldownEnabled(bool,uint8) (ARULSINGAM.sol#794-797) should emit an event for:
        - cooldownTimerInterval = _interval (ARULSINGAM.sol#796)
ARULSINGAM.setMaxTxPercent_base1000(uint256) (ARULSINGAM.sol#828-831) should emit an event for:
        - maxTxAmount = totalSupply().mul(amount).div(1000) (ARULSINGAM.sol#829)
ARULSINGAM.setMaxWalletPercent_base1000(uint256) (ARULSINGAM.sol#837-840) should emit an event for:
        - maxWalletAmount = totalSupply().mul(amount).div(1000) (ARULSINGAM.sol#838)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

ARULSINGAM.clearBNB(address).wallet (ARULSINGAM.sol#995) lacks a zero-check on :
                - (s) = wallet.call{value: balance}() (ARULSINGAM.sol#997)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has external calls inside a loop: sellPath[1] = uniswap
V2Router02.WETH() (ARULSINGAM.sol#634)
ARULSINGAM.forwardEther(address,uint256) (ARULSINGAM.sol#1004-1007) has external calls inside a loop: (os) = wallet.call{value
: amount}() (ARULSINGAM.sol#1005)
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has external calls inside a loop: ethValue = uniswapV2R
outer02.getAmountsOut(marketingTokens + devTokens + charityTokens,sellPath)[1] (ARULSINGAM.sol#677)
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has external calls inside a loop: uniswapV2Router02.swa
pExactTokensForETHSupportingFeeOnTransferTokens(toSell,0,sellPath,address(this),block.timestamp) (ARULSINGAM.sol#687-693)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
```

```
        - transfer(taxWallets[burn],burnTokens) (ARULSINGAM.sol#725)
                - (os) = wallet.call{value: amount}() (ARULSINGAM.sol#1005)
                - uniswapV2Router02.swapExactTokensForETHSupportingFeeOnTransferTokens(toSell,0,sellPath,address(this),block.t
imestamp) (ARULSINGAM.sol#687-693)
        External calls sending eth:
        - _transfer(from,address(this),tax) (ARULSINGAM.sol#664)
                - (os) = wallet.call{value: amount}() (ARULSINGAM.sol#1005)
        - forwardEther(taxWallets[marketing],marketingEth) (ARULSINGAM.sol#700)
                - (os) = wallet.call{value: amount}() (ARULSINGAM.sol#1005)
        - forwardEther(taxWallets[dev],devEth) (ARULSINGAM.sol#703)
                - (os) = wallet.call{value: amount}() (ARULSINGAM.sol#1005)
        - forwardEther(taxWallets[charity],charityEth) (ARULSINGAM.sol#706)
                - (os) = wallet.call{value: amount}() (ARULSINGAM.sol#1005)
        - transfer(taxWallets[burn],burnTokens) (ARULSINGAM.sol#709)
                - (os) = wallet.call{value: amount}() (ARULSINGAM.sol#1005)
        - transfer(taxWallets[marketing],marketingTokens) (ARULSINGAM.sol#715)
                - (os) = wallet.call{value: amount}() (ARULSINGAM.sol#1005)
        - transfer(taxWallets[dev],devTokens) (ARULSINGAM.sol#719)
                - (os) = wallet.call{value: amount}() (ARULSINGAM.sol#1005)
        - transfer(taxWallets[charity],charityTokens) (ARULSINGAM.sol#722)
                - (os) = wallet.call{value: amount}() (ARULSINGAM.sol#1005)
        - transfer(taxWallets[burn],burnTokens) (ARULSINGAM.sol#725)
                - (os) = wallet.call{value: amount}() (ARULSINGAM.sol#1005)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (ARULSINGAM.sol#277)
                - transfer(taxWallets[burn],burnTokens) (ARULSINGAM.sol#725)
        - Transfer(sender,recipient,amount) (ARULSINGAM.sol#234)
                - transfer(taxWallets[burn],burnTokens) (ARULSINGAM.sol#725)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(cooldownTimer[to] < block.timestamp,Please wait for cooldown between two buys) (ARULSINGAM.sol#
644)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has costly operations inside a loop:
        - marketingTokens += baseUnit * sellTaxes[marketing] (ARULSINGAM.sol#667)
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has costly operations inside a loop:
        - marketingTokens += baseUnit * buyTaxes[marketing] (ARULSINGAM.sol#653)
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has costly operations inside a loop:
        - devTokens += baseUnit * buyTaxes[dev] (ARULSINGAM.sol#654)
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has costly operations inside a loop:
        - devTokens += baseUnit * sellTaxes[dev] (ARULSINGAM.sol#668)
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has costly operations inside a loop:
        - charityTokens += baseUnit * sellTaxes[charity] (ARULSINGAM.sol#669)
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has costly operations inside a loop:
        - burnTokens += baseUnit * sellTaxes[burn] (ARULSINGAM.sol#670)
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has costly operations inside a loop:
        - marketingTokens = 0 (ARULSINGAM.sol#730)
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has costly operations inside a loop:
        - devTokens = 0 (ARULSINGAM.sol#731)
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has costly operations inside a loop:
        - charityTokens = 0 (ARULSINGAM.sol#732)
ARULSINGAM.handleTax(address,address,uint256) (ARULSINGAM.sol#631-746) has costly operations inside a loop:
        - burnTokens = 0 (ARULSINGAM.sol#733)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```

```
Pragma version0.8.4 (ARULSINGAM.sol#5) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in ARULSINGAM.clearBNB(address) (ARULSINGAM.sol#995-999):
        - (s) = wallet.call{value: balance}() (ARULSINGAM.sol#997)
Low level call in ARULSINGAM.forwardEther(address,uint256) (ARULSINGAM.sol#1004-1007):
        - (os) = wallet.call{value: amount}() (ARULSINGAM.sol#1005)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (ARULSINGAM.sol#346) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (ARULSINGAM.sol#347) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (ARULSINGAM.sol#364) is not in mixedCase
Function IUniswapV2Router01.WETH() (ARULSINGAM.sol#400) is not in mixedCase
Parameter ARULSINGAM.cooldownEnabled(bool,uint8)._status (ARULSINGAM.sol#794) is not in mixedCase
Parameter ARULSINGAM.cooldownEnabled(bool,uint8)._interval (ARULSINGAM.sol#794) is not in mixedCase
Function ARULSINGAM.setMaxTxPercent_base1000(uint256) (ARULSINGAM.sol#828-831) is not in mixedCase
Function ARULSINGAM.setMaxWalletPercent_base1000(uint256) (ARULSINGAM.sol#837-840) is not in mixedCase
Parameter ARULSINGAM.swapBackSettings(bool,uint256)._enabled (ARULSINGAM.sol#867) is not in mixedCase
Function ARULSINGAM.multiTransfer_fixed(address[],uint256) (ARULSINGAM.sol#966-971) 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 (ARUL
SINGAM.sol#405) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint
256).amountBDesired (ARULSINGAM.sol#406)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
```

```
ARULSINGAM.devTaxBuy (ARULSINGAM.sol#544) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
ARULSINGAM.marketingTaxBuy (ARULSINGAM.sol#545) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
ARULSINGAM.charityTaxBuy (ARULSINGAM.sol#546) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
ARULSINGAM.burnTaxBuy (ARULSINGAM.sol#547) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
ARULSINGAM.devTaxSell (ARULSINGAM.sol#549) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
ARULSINGAM.marketingTaxSell (ARULSINGAM.sol#550) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
ARULSINGAM.charityTaxSell (ARULSINGAM.sol#551) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
ARULSINGAM.burnTaxSell (ARULSINGAM.sol#552) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
ARULSINGAM.devTaxWallet (ARULSINGAM.sol#557) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
ARULSINGAM.marketingTaxWallet (ARULSINGAM.sol#558) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
ARULSINGAM.charityTaxWallet (ARULSINGAM.sol#559) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
ARULSINGAM.burnTaxWallet (ARULSINGAM.sol#560) is never used in ARULSINGAM (ARULSINGAM.sol#534-1038)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

ARULSINGAM.burnTaxBuy (ARULSINGAM.sol#547) should be constant
ARULSINGAM.burnTaxSell (ARULSINGAM.sol#552) should be constant
ARULSINGAM.burnTaxWallet (ARULSINGAM.sol#560) should be constant
ARULSINGAM.charityTaxBuy (ARULSINGAM.sol#546) should be constant
ARULSINGAM.charityTaxSell (ARULSINGAM.sol#551) should be constant
ARULSINGAM.charityTaxWallet (ARULSINGAM.sol#559) should be constant
ARULSINGAM.denominator (ARULSINGAM.sol#540) should be constant
ARULSINGAM.devTaxBuy (ARULSINGAM.sol#544) should be constant
ARULSINGAM.devTaxSell (ARULSINGAM.sol#549) should be constant
ARULSINGAM.devTaxWallet (ARULSINGAM.sol#557) should be constant
ARULSINGAM.marketingTaxBuy (ARULSINGAM.sol#545) should be constant
ARULSINGAM.marketingTaxSell (ARULSINGAM.sol#550) should be constant
ARULSINGAM.marketingTaxWallet (ARULSINGAM.sol#558) should be constant
ARULSINGAM.taxStatus (ARULSINGAM.sol#575) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

```
ARULSINGAM.maxSupply (ARULSINGAM.sol#538) should be immutable
ARULSINGAM.uniswapV2Factory (ARULSINGAM.sol#587) should be immutable
ARULSINGAM.uniswapV2Pair (ARULSINGAM.sol#588) should be immutable
ARULSINGAM.uniswapV2Router02 (ARULSINGAM.sol#586) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
ARULSINGAM.sol analyzed (11 contracts with 84 detectors), 118 result(s) found
```

Solidity Static Analysis

ARULSINGAM.sol

Security

Check-effects-interaction:

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

more

Pos: 631:4:

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: 652:36:

Gas & Economy

Gas costs:

Gas requirement of function ARULSINGAM.multiTransfer_fixed 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: 966:4:

Gas costs:

Gas requirement of function ARULSINGAM.multiTransfer 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: 952:4:

Email: audit@EtherAuthority.io

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: 968:8:

ERC

ERC20:

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

more

Pos: 337:4:

Miscellaneous

Constant/View/Pure functions:

IUniswapV2Router02.swapExactTokensForETHSupportingFeeOnTransferTokens(ui

: Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 524:4:

No return:

IUniswapV2Router02.removeLiquidityETHWithPermitSupportingFeeOnTransferTok

Defines a return type but never explicitly returns a value.

Pos: 501:4:

Similar variable names:

ARULSINGAM.clearBNB(address) : Variables have very similar names "_balances" and "balance". Note: Modifiers are currently not considered by this static analysis.
Pos: 997:39:

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: 1031:8:

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: 821:32:

Solhint Linter

ARULSINGAM.sol

```
ARULSINGAM.sol:11:18: Error: Parse error: missing ';' at '{'
ARULSINGAM.sol:19:18: Error: Parse error: missing ';' at '{'
ARULSINGAM.sol:26:18: Error: Parse error: missing ';' at '{'
ARULSINGAM.sol:36:18: Error: Parse error: missing ';' at '{'
ARULSINGAM.sol:43:18: Error: Parse error: missing ';' at '{'
ARULSINGAM.sol:74:18: Error: Parse error: missing ';' at '{'
ARULSINGAM.sol:85:18: Error: Parse error: missing ';' at '{'
ARULSINGAM.sol:96:18: Error: Parse error: missing ';' at '{'
ARULSINGAM.sol:193:18: Error: Parse error: missing ';' at '{'
ARULSINGAM.sol:210:18: Error: Parse error: missing ';' at '{'
ARULSINGAM.sol:229:18: Error: Parse error: missing ';' at '{'
ARULSINGAM.sol:258:18: 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