Rodeo Solutions

Develop - Audit - Coach

ROLiToken Smart Contract Audit

Rodeo Solutions

June 2021

Commision

Audited Project	ROLiToken
Project website	https://rolitoken.com/
Contract Owner	0×78CAEDF21b765A43b8e419B3E15797d98A148322
SmartContract Address	0×53e40cf9a180A5951905F8210300ed16cd0C5cB8
Blockchain	Binance Main Smart Chain

Rodeo Solutions was commissioned by ROLiToken owners to perform an audit of their main smart contract.

The purpose of the audit was to achieve the following:

- Ensure that the smart contract functions as intended.
- Identify potential security issues with the smart contract.

The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Rodeo Solutions and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Rodeo Solutions) owe no duty of care towards you or any other person, nor does Rodeo Solutions make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Rodeo Solutions hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Rodeo Solutions hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Rodeo Solutions, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

$ROLI Properties

Contract name	RoliToken
Contract address	0×53e40cf9a180A5951905F8210300ed16cd0C5cB8
Total supply	1000T
Token ticker	ROLI
Decimals	9
Token holders	1
Transactions count	1
Top 100 holders dominance	100.00%
Liquidity fee	4%
Tax fee	4%
Total fees	8%
Mintable	No
Burnable	No
Uniswap V2 pair	0×d855184236f505cef4d9d0d5cd25d9d471df32eb
Contract deployer address	0×78CAEDF21b765A43b8e419B3E15797d98A148322
Contract’s current owner address	0×78CAEDF21b765A43b8e419B3E15797d98A148322

As of 16/06/2021

Contract Functions

Public

View

```
name()
symbol()
decimals()
totalSupply()
transfer(address recipient, uint256 amount)
allowance(address owner, address spender)
approve(address spender, uint256 amount)
isExcludedFromReward(address account)
totalFees()
reflectionFromToken(uint256 tAmount, bool deductTransferFee)
tokenFromReflection(uint256 rAmount)
```

Virtual

```
increaseAllowance(address spender, uint256 addedValue)
decreaseAllowance(address spender, uint256 subtractedValue)
```

Executables

```
balanceOf(address account)
transferFrom(address sender, address recipient, uint256 amount)
```

Owner Executables

```
excludeFromReward(address account)
excludeFromFee(address account)
includeInFee(address account)
setTaxFeePercent(uint256 taxFee)
setLiquidityFeePercent(uint256 liquidityFee)
setMaxTxPercent(uint256 maxTxPercent)
setSwapAndLiquifyEnabled(bool _enabled)
setTaxEnable(bool _enable)
Extra standard from Ownable Contract
```

Checklist

Compiler errors.	Passed
Possible delays in data delivery.	Passed
Timestamp dependence.	Passed
Integer Overflow and Underflow.	Passed
DoS with Revert.	Passed
DoS with block gas limit.	Low issues
Methods execution permissions.	Passed
Economy model of the contract.	Passed
Private user data leaks.	Passed
Malicious Event log.	Passed
Scoping and Declarations.	Passed
Uninitialized storage pointers.	Passed
Arithmetic accuracy.	Passed
Design Logic.	Not Passed
Cross-function race conditions.	Passed
Fallback function security.	Passed
Safe Open Zeppelin contracts implementation and usage.	Not Passed

Potential Issues

Main concern

- 50% of the supply should've been burned on deployment.

As stated above the token is not burnable by definition meaning the standard ERC20/IERC20 *_burn* function is not present in the code. In order to have "Manual Burns" as the website states the following code should be incorporated.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Burnable.sol

In this case what the dev did was that, at the moment of creation when all the properties are defined, he divided the initial supply by 2 and added the result to the Owner address instead of the total supply (1000T) and then burning.

```
uint256 half = _rTotal.div(2);
_rOwned[_msgSender()] = half;
_rOwned[address(0)] = half;
```

This method wasn't implemented correctly either as the burned amount wasn't deducted from the total supply. A couple of additional variables should be added too to show the burned amount and the initial supply.

There are a couple of issues with this method:

- First of all this is not a standard practise so any investor who looks closer at the code will see something not usual happening. This piece of code is not commented on the code either so the objective is not clear just by reading those lines.
- The burnage is not reflected on the totalSupply of the token nor a transaction registered on the blockchain.
- There are no "initial supply" and "burned" variables.
- No additional burns can be done

Several transfers on deployment

As soon as the contract is deployed several token transfers happen, adding $ROLI to several addresses.

```
_transfer(_msgSender(), 0x32E9F2140B1018306bAA5Db079e2cF074b827EfF, 500000 * 10**6 * 10**9);
_transfer(_msgSender(), 0xBeE7Fb6C80B2BCeBfABF5D86D8618bBaE1753B06, 500000 * 10**6 * 10**9);
_transfer(_msgSender(), 0xf46916544CEfDB426fB01cBC5Ee843f75FABDb63, 500000 * 10**6 * 10**9);
_transfer(_msgSender(), 0xcc5fBDd79533Cf0F8680A180477bdD72ef4FE0C8, 500000 * 10**6 * 10**9);
_transfer(_msgSender(), 0x3391db3d80D0570F0f2251c9968c8C6b4BB41AC6, 500000 * 10**6 * 10**9);
_transfer(_msgSender(), 0x1FA0df5341b9b4EF7849e55CBCf4cCd5a6462Ec5, 500000 * 10**6 * 10**9);
_transfer(_msgSender(), 0x6c852Ba3352273E85CB02554e13f9ff236a4911F, 500000 * 10**6 * 10**9);
_transfer(_msgSender(), 0x21218da28460CCb85838197F7052f6374D19A035, 500000 * 10**6 * 10**9);
_transfer(_msgSender(), 0x248564dF4F3C62Dc91798B8549d8dfD0B8F25788, 500000 * 10**6 * 10**9);
_transfer(_msgSender(), 0xA9053c4DD477A68cDd171cD901C5e0420647e52b, 500000 * 10**6 * 10**9);

_transfer(_msgSender(), 0x70C5103F473929419723B582a22EC8782E009cde, 5000000 * 10**6 * 10**9);

_transfer(_msgSender(), 0x0e3183E4E68d427260FBF7ADd4E827e47E903CD6, 1000000 * 10**6 * 10**9);
_transfer(_msgSender(), 0x0e3183E4E68d427260FBF7ADd4E827e47E903CD6, 1000000 * 10**6 * 10**9);

_transfer(_msgSender(), 0x1E551ceF984032C69990C3F8F60B3F2690Aba740, 1000000 * 10**6 * 10**9);
```

If this was requested as an initial transfer for investors the addresses and amounts should be verified. Otherwise this should be deleted from the code.

This section can be related to “make up dev/admin tokens 13T total dispersed prior to DXsale”, nonetheless, this is not clear in the code and could raise suspicions.

Transfer addresses:

0×32E9F2140B1018306bAA5Db079e2cF074b827EfF
0×BeE7Fb6C80B2BCeBfABF5D86D8618bBaE1753B06
0×f46916544CEfDB426fB01cBC5Ee843f75FABDb63
0×cc5fBDd79533Cf0F8680A180477bdD72ef4FE0C8
0×3391db3d80D0570F0f2251c9968c8C6b4BB41AC6
0×1FA0df5341b9b4EF7849e55CBCf4cCd5a6462Ec5
0×6c852Ba3352273E85CB02554e13f9ff236a4911F
0×21218da28460CCb85838197F7052f6374D19A035
0×248564dF4F3C62Dc91798B8549d8dfD0B8F25788
0×A9053c4DD477A68cDd171cD901C5e0420647e52b
0×70C5103F473929419723B582a22EC8782E009cde
0×0e3183E4E68d427260FBF7ADd4E827e47E903CD6
0×0e3183E4E68d427260FBF7ADd4E827e47E903CD6
0×1E551ceF984032C69990C3F8F60B3F2690Aba740

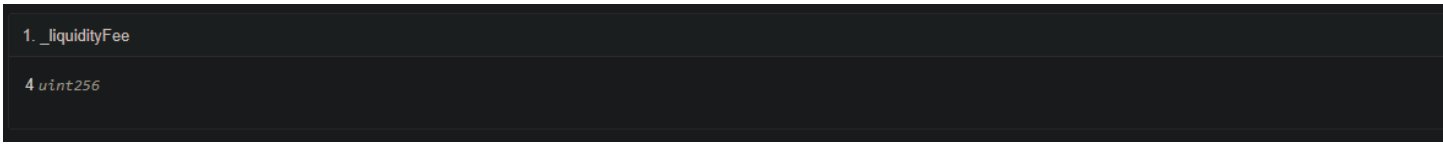
Taxes are not clear for potential investors

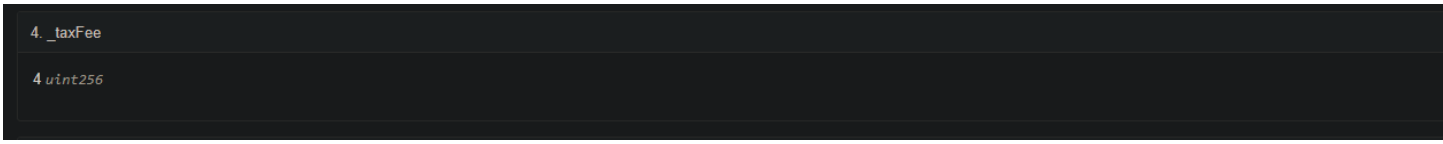
When verifying the taxes it's clear that there are 8% taken from every transaction:

```
uint256 public _taxFee = 4;
uint256 private _previousTaxFee = _taxFee;

uint256 public _liquidityFee = 4;
uint256 private _previousLiquidityFee = _liquidityFee;
```

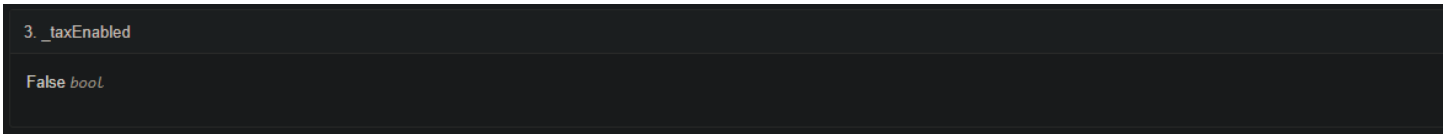
The 8% is also clear on BSCScan:





The issue rises when checking if the taxes are enabled. This is set to false, meaning that currently there are no taxes being taken:

```
bool public _taxEnabled = false;
```



The logic in the transfer function used to send or receive tokens is not clear at first sight which is not very well received by potential investors.

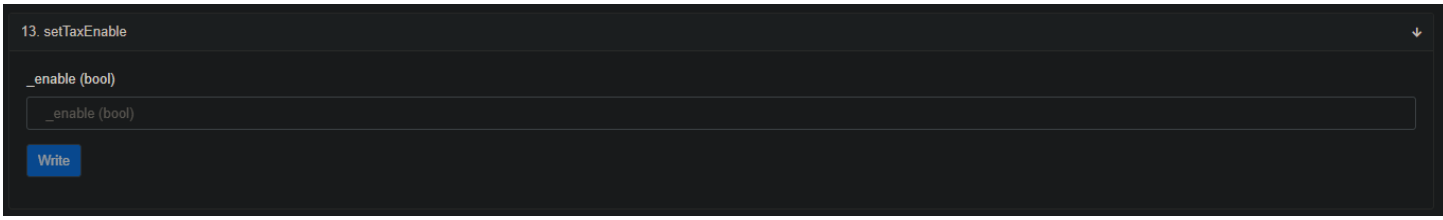
```
//indicates if fee should be deducted from transfer
bool takeFee = true;

//if any account belongs to _isExcludedFromFee account then remove the fee
if(!_taxEnabled || _isExcludedFromFee[from] || _isExcludedFromFee[to]){
    takeFee = false;
}

//transfer amount, it will take tax, burn, liquidity fee
_tokenTransfer(from,to,amount,takeFee);
```

Here it sets a temporal "takeFee" variable as true but if it finds that _taxEnabled is false, it will set that temporal variable to false. Making it indeed not take any taxes from the transactions.

Nonetheless there's a function to enable the taxes so changing the code is not a must.



Owner privileges

- The owner can Exclude or Add an address from the rewards program. Not available on Roli.

```
function excludeFromReward(address account) public onlyOwner() {
    // require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not exclude Uniswap router.');
    require(!_isExcluded[account], "Account is already excluded");
    if(_rOwned[account] > 0) {
        _tOwned[account] = tokenFromReflection(_rOwned[account]);
    }
    _isExcluded[account] = true;
    _excluded.push(account);
}

function includeInReward(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

- The owner can Exclude or Add an address from being taxed.

```
function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}

function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}
```

- The owner can change the Tax percentages at any time.

```
function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
    _taxFee = taxFee;
    emit SetTaxFeePercent(taxFee);
}

function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {
    _liquidityFee = liquidityFee;
    emit SetLiquidityFeePercent(liquidityFee);
}
```

- The owner can enable or disable taxes at any time.

```
function setTaxEnable(bool _enable) public onlyOwner {
    _taxEnabled = _enable;
    emit SetTaxEnable(_enable);
}
```

- The owner can toggle the automatic liquidity swap at any time.

```
function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
    swapAndLiquifyEnabled = _enabled;
    emit SwapAndLiquifyEnabledUpdated(_enabled);
}
```

- The owner can set the maximum amount of a transaction at any time.

```
function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(
        10**2
    );
    emit SetMaxTxPercent(maxTxPercent);
}
```

- Additionally the owner can execute the standard functions related to the Ownable Library

Conclusion

As far as the rest of the code goes, it uses the industry standards taking most of the OpenZeppelin and SafeMath standards. There are not any further concerns and can be verified against the code provided officially in https://github.com/OpenZeppelin/openzeppelin-contracts

The code was deployed on the Binance TestNet in order to verify transfer funds from excluded and from not excluded from fees and there are no issues.

Neither in the regular Read functions or Write functions.

The code should have some changes done to it being the following in order of importance:

1. Incorporate the following code https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Burnable.sol
2. Burn 50% of the initial supply at the beginning and adding that to the owner address
3. initialSupply variable
4. burnedAmount variable
5. Verify multiple transfers to addresses
6. Change the _taxEnabled variable to True if the contract gets redeployed. Otherwise use the corresponding function to set it to True.
7. Further comments explaining the non-standard logic