



AUDIT REPORT

SecureWise

MULTIRUSH (MRT)



MULTIRUSH



Contents

- 
- 02 Disclaimer
 - 03 Overview
 - 04 Quick Result
 - 05 Auditing Approach and Methodologies
 - 06 Automated Analysis
 - 08 Inheritance Graph
 - 10 Manual Review

Disclaimer

SecureWise provides the smart contract audit of solidity. Audit and report are for informational purposes only and not, nor should be considered, as an endorsement to engage with, invest in, participate, provide an incentive, or disapprove, criticise, discourage, or purport to provide an opinion on any particular project or team.

This audit report doesn't provide any warranty or guarantee regarding the nature of the technology analysed. These reports, in no way, provide investment advice, nor should be used as investment advice of any sort. Investors must always do their own research and manage their risk.

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 SecureWise and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) SecureWise owe no duty of care towards you or any other person, nor does SecureWise 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 SecureWise 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, SecureWise hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against SecureWise, 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.

Overview

Token Name: MultiRush (**MRT**)

Methodology: Automated Analysis, Manual Code Review

Language: Solidity

Contract Address: 0x63611Dad015f89615515Eb042EAf37a2cDd986BB

ContractLink: <https://bscscan.com/address/0x63611Dad015f89615515Eb042EAf37a2cDd986BB>

Network: Binance Smart Chain (**BSC**)

Supply: 1000000000

Website: <https://multirush.org/>






Twitter: <https://twitter.com/multirushapp>

Telegram: <https://t.me/multirush>

Report Date: May 16, 2023

Quick Result

SecureWise has applied the automated and manual analysis of Smart Contract and were reviewed for common contract vulnerabilities and centralized exploits

	The owner can exclude accounts from rewards
	The owner can exclude accounts from fees
	The owner can set fees with limit up to 6%
	The owner can change swap settings
	The owner can withdraw token(except native token) from the contract

Page 10 for more details

MultiRush (MRT) as succesfully **PASSED** the smart contract audit with **LOW** severity issue

Auditing Approach and Methodologies

SecureWise has performed starting with analyzing the code, issues, code quality, and libraries. Reviewed line-by-line by our team. Finding any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

Methodology

- Understanding the size, scope and functionality of your project's source code
- Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Testing and automated analysis of the Smart Contract to determine proper logic has been followed throughout the whole process
- Deploying the code on testnet using multiple live test
- Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.
- Checking whether all the libraries used in the code are on the latest version.

Goals

Smart Contract System is secure, resilient and working according to the specifications and without any vulnerabilities.

Risk Classification

High: Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, of the contract and its functions. Must be fixed as soon as possible.

Medium: Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Must be fixed as soon as possible.

Low: Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.

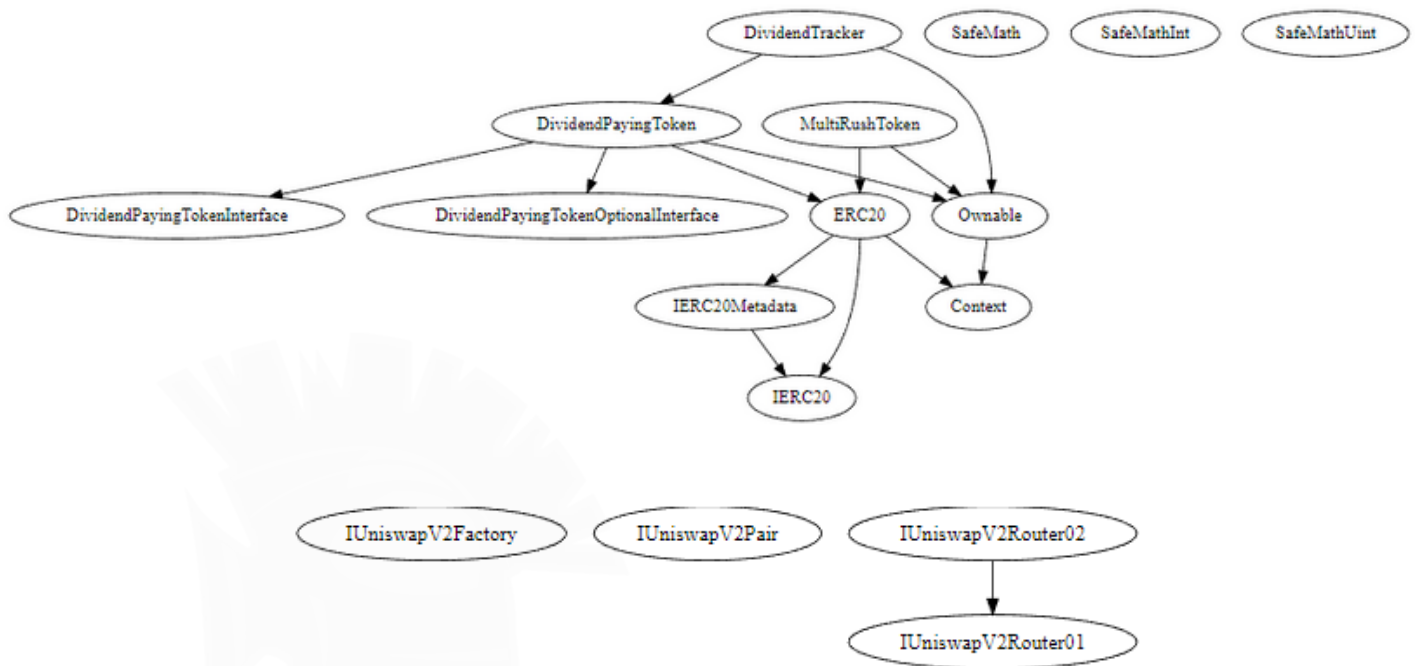
Automated Analysis

DividendPayingTokenInterface	Interface			
L	dividendOf	External !		NO !
DividendPayingTokenOptionalInterface	Interface			
L	withdrawableDividendOf	External !		NO !
L	withdrawnDividendOf	External !		NO !
L	accumulativeDividendOf	External !		NO !
DividendPayingToken	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
L		Public !	●	ERC20
L		External !	■	NO !
L	distributeDividends	Public !	●	onlyOwner
L	_withdrawDividendOfUser	Internal !	●	
L	dividendOf	Public !		NO !
L	withdrawableDividendOf	Public !		NO !
L	withdrawnDividendOf	Public !		NO !
L	accumulativeDividendOf	Public !		NO !
L	_transfer	Internal !	●	
L	_mint	Internal !	●	
L	_burn	Internal !	●	
L	_setBalance	Internal !	●	
DividendTracker	Implementation	Ownable, DividendPayingToken		
L		Public !	●	DividendPayingToken
L	_transfer	Internal !		
L	updateMinimumTokenBalanceForDividends	External !	●	onlyOwner
L	excludeFromDividends	External !	●	onlyOwner
L	updateClaimWait	External !	●	onlyOwner
L	getAccount	Public !		NO !
L	setBalance	External !	●	onlyOwner
L	processAccount	Public !	●	onlyOwner
MultiRushToken	Implementation	ERC20, Ownable		
L		Public !	●	ERC20
L		External !	■	NO !
L	claimStuckTokens	External !	●	onlyOwner
L	addRewardToken	Public !	●	onlyOwner
L	removeRewardToken	Public !	●	onlyOwner
L	isContract	Internal !		

Automated Analysis

L	sendBNB	Internal 🔒	●	
L	_setAutomatedMarketMakerPair	Private 🔒	●	
L	excludeFromFees	External !	●	onlyOwner
L	isExcludedFromFees	Public !		NO !
L	updateBuyFees	External !	●	onlyOwner
L	updateSellFees	External !	●	onlyOwner
L	changeDevWallet	External !	●	onlyOwner
L	changeMarketing	External !	●	onlyOwner
L	_transfer	Internal 🔒	●	
L	setSwapEnabled	External !	●	onlyOwner
L	setSwapTokensAtAmount	External !	●	onlyOwner
L	setSwapWithLimit	External !	●	onlyOwner
L	swapAndSendDividends	Private 🔒	●	
L	updateDividendTracker	Public !	●	onlyOwner
L	updateGasForProcessing	Public !	●	onlyOwner
L	updateMinimumBalanceForDividends	External !	●	onlyOwner
L	updateClaimWait	External !	●	onlyOwner
L	getClaimWait	External !		NO !
L	excludeFromDividends	External !	●	onlyOwner
L	getAccountDividendsInfo	External !		NO !
L	getAmountsOut	Public !		NO !
L	getDashboard	External !		NO !
L	tokenList	External !		NO !
L	claim	External !	●	NO !
L	claimAddress	External !	●	onlyOwner

Inheritance Graph



Components

Contracts	Libraries	Interfaces	Abstract
4	3	8	2

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

Public	Payable
124	6

External	Internal	Private	Pure	View
97	112	2	26	51

StateVariables

Total	Public
43	26

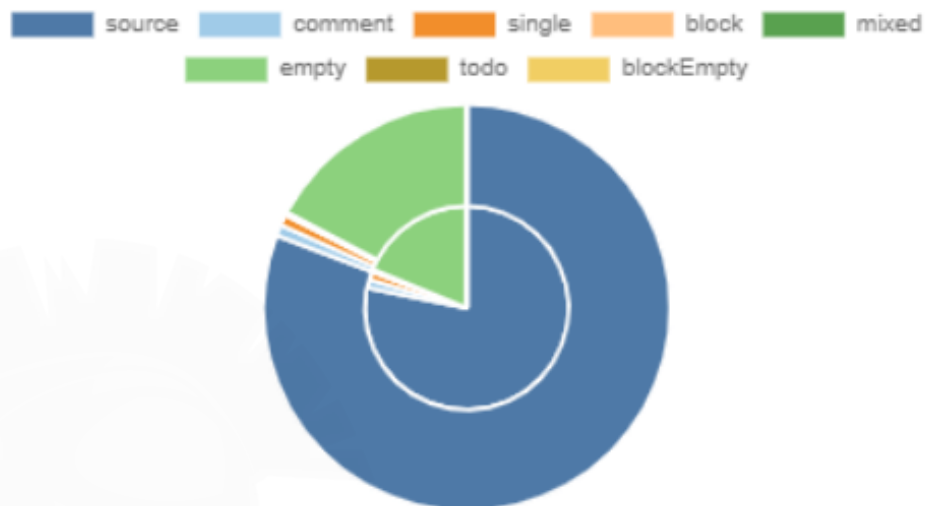
Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
0.8.19		yes		

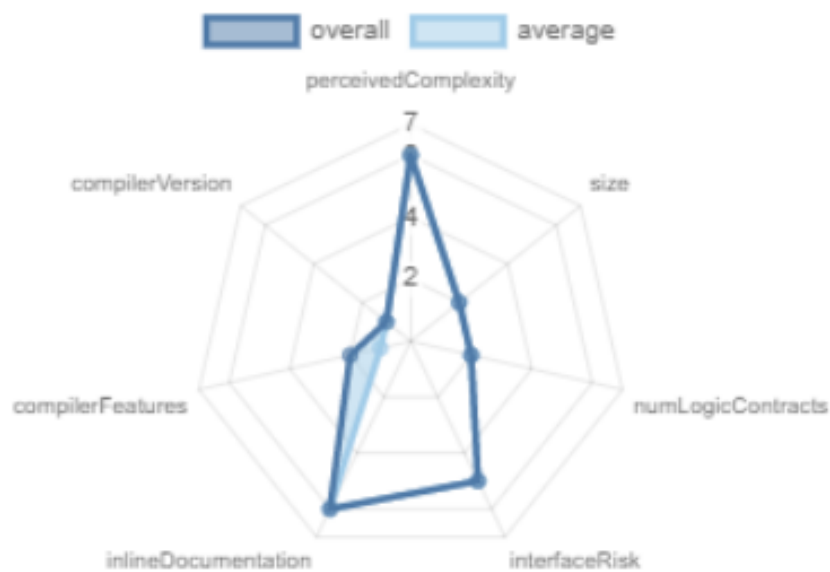
Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
yes					yes → NewContract:DividendTracker

TryCatch	Σ Unchecked
yes	

Source Lines



Risk



Manual Review

Owner can exclude addresses from rewards

```
function excludeFromDividends(address account) external onlyOwner {  
    require(!excludedFromDividends[account]);  
    excludedFromDividends[account] = true;  
  
    _setBalance(account, 0);  
  
    emit ExcludeFromDividends(account);  
}
```

Description

Function of **excludeFromReward** in a smart contract which allows the owner to exclude an account from receiving rewards.

Recommendation

it is recommended to use a proper access control mechanism. You should carefully manage the private key of the owner's account. You should use powerful security mechanism that will prevent a single user from accessing the contract owner functions.

Manual Review

Owner can exclude/include addresses from fees

```
function excludeFromFees(address account) external onlyOwner {  
    require(!_isExcludedFromFees[account], "Account is already the value of 'excluded'");  
    _isExcludedFromFees[account] = true;  
  
    emit ExcludeFromFees(account, true);  
}
```

Description

Authorizing privileged roles to exclude accounts from fees. After excluding the user from accounts, the user trades without paying a any fee and the other user sees it). But may apply in some cases like (owner wallets, contract, marketing-treasury wallets etc..)

Recommendation

You should carefully manage the private key of the owner's account. You should use powerful security mechanism that will prevent a single user from accessing the contract owner functions. That risk can be prevented by temporarily locking the contract or renouncing ownership

Manual Review

Owner can change total buy/sell fee not greater than 6%

```
function updateBuyFees(uint256 _DevFeeOnBuy,uint256 _MarketingFeeOnBuy,uint256 _TreasuryFeeOnBuy) external onlyOwner {
    DevFeeOnBuy = _DevFeeOnBuy;
    TreasuryFeeOnBuy = _TreasuryFeeOnBuy;
    MarketingFeeOnBuy = _MarketingFeeOnBuy;
    _totalFeesOnBuy = DevFeeOnBuy + MarketingFeeOnBuy + TreasuryFeeOnBuy;

    require(_totalFeesOnBuy <= 6,"Buy Fees must be less than 6%");
    emit UpdateBuyFees(_DevFeeOnBuy, _MarketingFeeOnBuy, _TreasuryFeeOnBuy);
}
```

```
function updateSellFees(uint256 _DevFeeOnSell,uint256 _MarketingFeeOnSell,uint256 _TreasuryFeeOnSell) external onlyOwner {
    DevFeeOnSell = _DevFeeOnSell;
    TreasuryFeeOnSell = _TreasuryFeeOnSell;
    MarketingFeeOnSell = _MarketingFeeOnSell;
    _totalFeesOnSell =DevFeeOnSell +MarketingFeeOnSell +TreasuryFeeOnSell;

    require(_totalFeesOnSell <= 6,"Sell Fees must be less than 6%");
    emit UpdateSellFees(_DevFeeOnSell,_MarketingFeeOnSell,_TreasuryFeeOnSell);
}
```

Description

The functions updates the fees associated with selling and buying a particular token. **Buy/Sell fees do not exceed 6% of the total**

Recommendation

it is recommended to use a proper access control mechanism. You should carefully manage the private key of the owner's account. You should use powerful security mechanism that will prevent a single user from accessing the contract owner functions.

Manual Review

Owner can update swap tokens at amount with an amount greater than 0

```
function setSwapTokensAtAmount(uint256 newAmount) external onlyOwner {  
    require(newAmount > totalSupply() / 1_000_000, "New Amount must more than 0.0001% of total supply");  
    swapTokensAtAmount = newAmount;  
}
```

Description

setSwapTokensAtAmount function updates the minimum token balance required in the contract for a transaction to take fees.

Recommendation

swapTokensAtAmount variable is set to a large value, it means that the contract will trigger the swap functionality when it holds a large amount of tokens, which can cause price volatility. On the other hand, setting **swapTokensAtAmount** to a very small value can lead to frequent swaps, which can increase gas fees and negatively impact the overall performance of the contract. It's important to strike a balance between these factors when choosing the value for **swapTokensAtAmount**

Manual Review

Owner can change swap settings

```
function setSwapEnabled(bool _swapEnabled) external onlyOwner {  
    require(  
        swapEnabled != _swapEnabled,  
        "Swap is already set to that state"  
    );  
    swapEnabled = _swapEnabled;  
}
```

```
function setSwapWithLimit(bool _swapWithLimit) external onlyOwner {  
    require(  
        swapWithLimit != _swapWithLimit,  
        "Swap with limit is already set to that state"  
    );  
    swapWithLimit = _swapWithLimit;  
}
```

Description

swapWithLimit variable to control whether swapping tokens should be performed with a limit. The function checks if the desired state (**_swapWithLimit**) is different from the current state. If they are different, the contract updates the **swapWithLimit** variable to the desired value. **swapEnabled** variable to control the overall swapping functionality of the contract. These functions provide the contract owner with the ability to toggle the swapping and limit functionality. By changing these variables, the contract's behavior regarding token swapping can be controlled.

Recommendation

Ensure that only the contract owner can execute these functions. It is recommended to use a proper access control mechanism. You should carefully manage the private key of the owner's account. You should use powerful security mechanism that will prevent a single user from accessing the contract owner functions.

Manual Review

The owner can withdraw token(except native token) from the contract

```
function claimStuckTokens(address token) external onlyOwner {
    require(token != address(this), "Owner cannot claim native tokens");
    if (token == address(0x0)) {
        payable(msg.sender).transfer(address(this).balance);
        return;
    }
    IERC20 ERC20token = IERC20(token);
    uint256 balance = ERC20token.balanceOf(address(this));
    ERC20token.transfer(msg.sender, balance);
}
```

Description

claimStuckTokens that allows the contract owner to claim tokens that are stuck in the contract. **require(token != address(this), "Owner cannot claim native tokens");** line ensures that the owner cannot claim the native tokens of the contract itself. It prevents accidental or intentional withdrawal of the contract's native tokens.

Recommendation

Ensure that only the contract owner can execute these functions. It is recommended to use a proper access control mechanism. You should carefully manage the private key of the owner's account. You should use powerful security mechanism that will prevent a single user from accessing the contract owner functions.



AUDIT REPORT

SecureWise



<https://securewise.info/>



<https://t.me/securewisehub>



<https://github.com/securewise>

