AUDIT REPORT

Secure Wise

GPULTIMATE (GPU)

https://securewise.info/ https://t.me/securewisehub https://github.com/securewise

Contents

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: GPUltimate **(GPU)**

Methodology: Automated Analysis, Manual Code Review

Language: Solidity

Contract Address: -

ContractLink: -

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

Supply: -

Website: https://gempad.app

Twitter: https://twitter.com/thegempad

Telegram: https://t.me/TheGemPad

Report Date: May 4, 2023

Owner Privileges

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

Owner can include/exclude addresses from rewards
Owner can change total buy/sell fee not greater than 30%
Owner can exclude/include addresses from fees
Owner can update max wallet amount greater than 0
Owner can update max tx amount with an amount greater than 0
Owner can exclude/include addresses from max transaction amount
Owner can change marketing wallet address
Owner can change treasury wallet address
Owner can change UniswapV2RouterPair address
Owner can change UniswapV2Router address
Owner can change automated market maker pair address and status
Owner can update swap tokens at amount with an amount greater than 0

Page 13 for more details

GPUltimate (GPU) has 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.

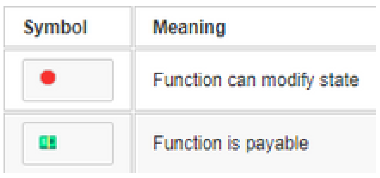
Risk Classification

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.

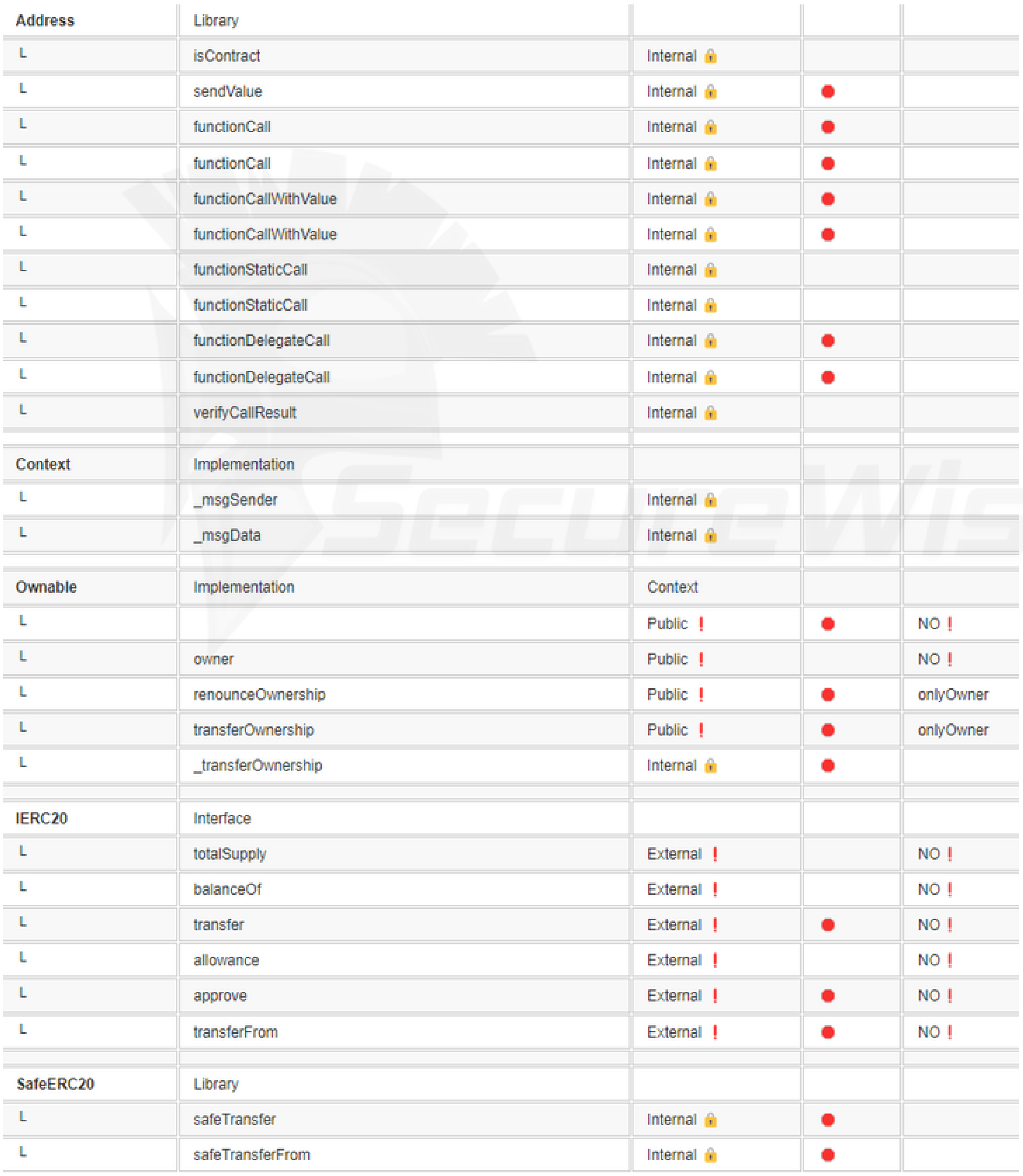
Medium: Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Must be fxed as soon as possible.

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

Automated Analysis

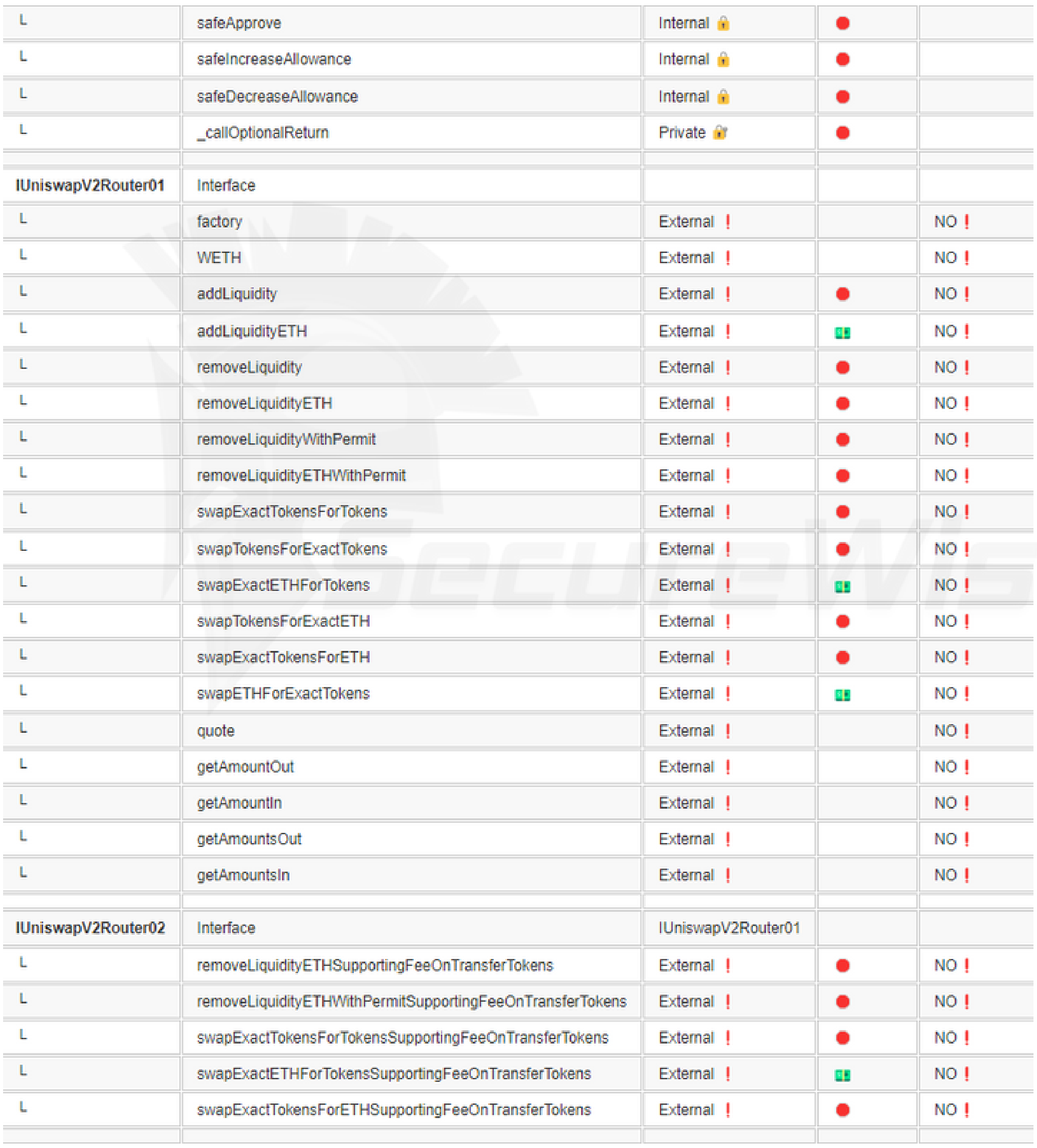


Symbol	Meaning
●	Function can modify state
💵	Function is payable



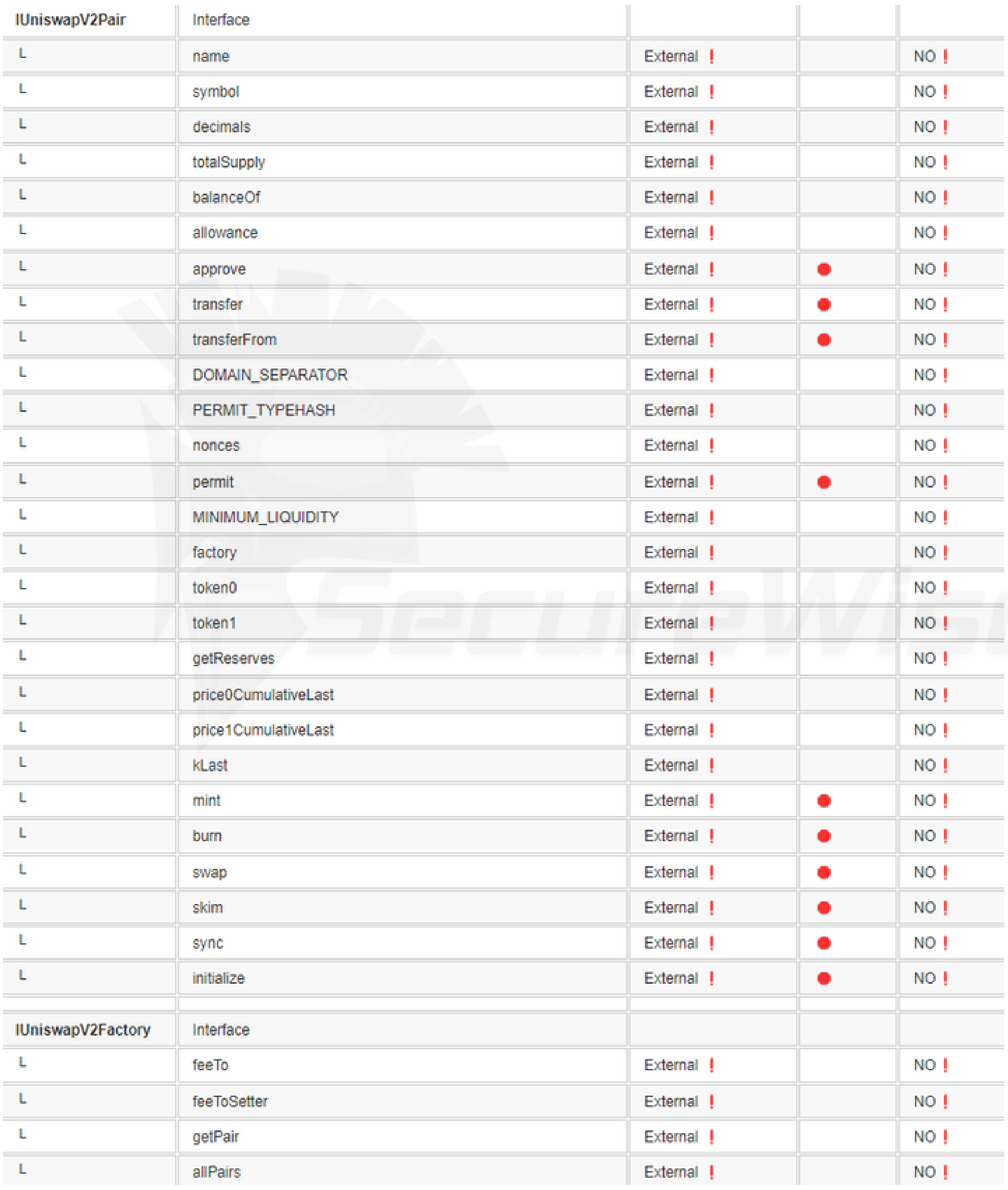
Address	Library			
L	isContract	Internal 🔒		
L	sendValue	Internal 🔒	●	
L	functionCall	Internal 🔒	●	
L	functionCall	Internal 🔒	●	
L	functionCallWithValue	Internal 🔒	●	
L	functionCallWithValue	Internal 🔒	●	
L	functionStaticCall	Internal 🔒		
L	functionStaticCall	Internal 🔒		
L	functionDelegateCall	Internal 🔒	●	
L	functionDelegateCall	Internal 🔒	●	
L	verifyCallResult	Internal 🔒		
Context	Implementation			
L	_msgSender	Internal 🔒		
L	_msgData	Internal 🔒		
Ownable	Implementation	Context		
L		Public !	●	NO !
L	owner	Public !		NO !
L	renounceOwnership	Public !	●	onlyOwner
L	transferOwnership	Public !	●	onlyOwner
L	_transferOwnership	Internal 🔒	●	
IERC20	Interface			
L	totalSupply	External !		NO !
L	balanceOf	External !		NO !
L	transfer	External !	●	NO !
L	allowance	External !		NO !
L	approve	External !	●	NO !
L	transferFrom	External !	●	NO !
SafeERC20	Library			
L	safeTransfer	Internal 🔒	●	
L	safeTransferFrom	Internal 🔒	●	

Automated Analysis



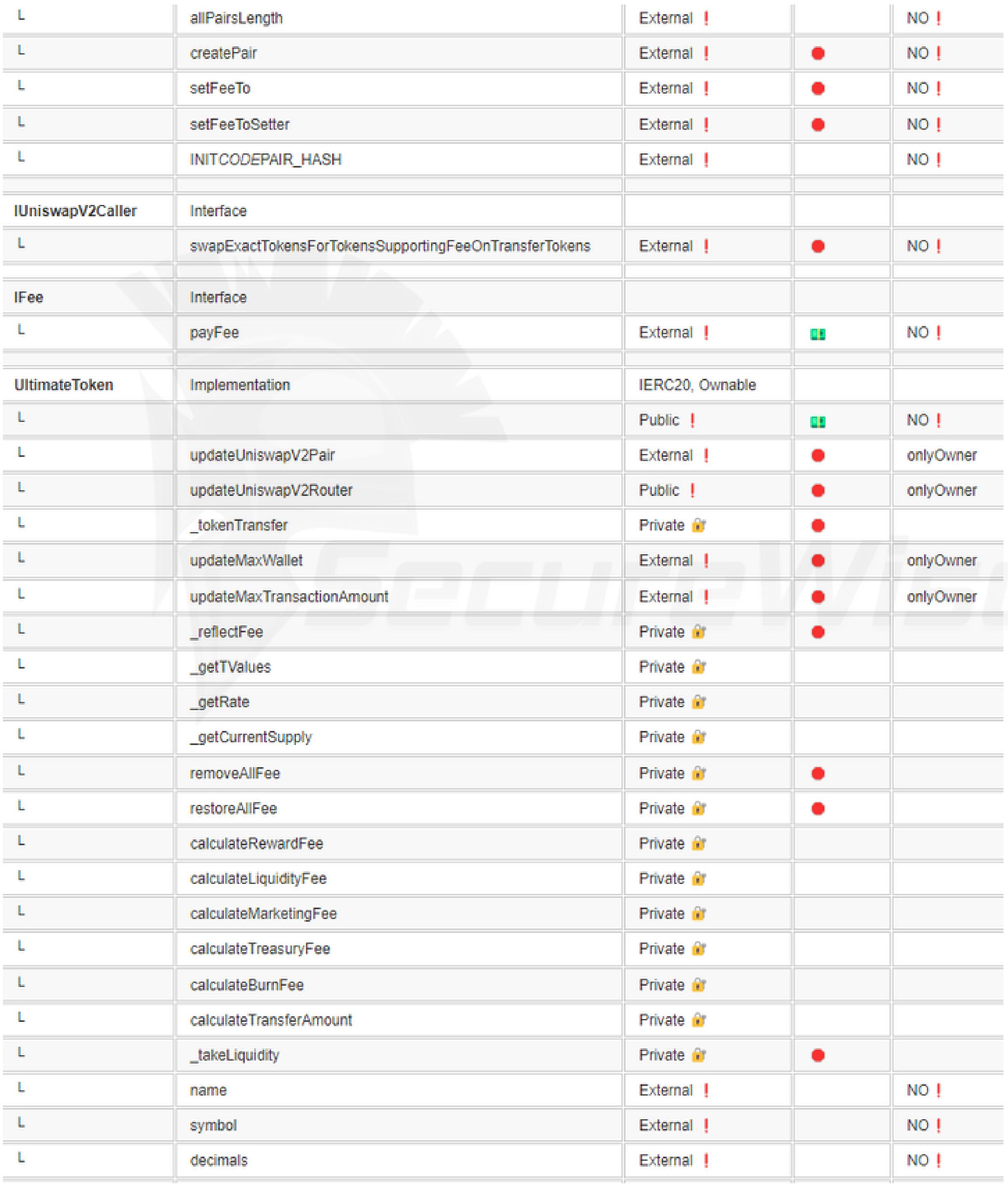
└	safeApprove	Internal 🔒	●	
└	safeIncreaseAllowance	Internal 🔒	●	
└	safeDecreaseAllowance	Internal 🔒	●	
└	_callOptionalReturn	Private 🔐	●	
IUniswapV2Router01	Interface			
└	factory	External ❗		NO ❗
└	WETH	External ❗		NO ❗
└	addLiquidity	External ❗	●	NO ❗
└	addLiquidityETH	External ❗	💵	NO ❗
└	removeLiquidity	External ❗	●	NO ❗
└	removeLiquidityETH	External ❗	●	NO ❗
└	removeLiquidityWithPermit	External ❗	●	NO ❗
└	removeLiquidityETHWithPermit	External ❗	●	NO ❗
└	swapExactTokensForTokens	External ❗	●	NO ❗
└	swapTokensForExactTokens	External ❗	●	NO ❗
└	swapExactETHForTokens	External ❗	💵	NO ❗
└	swapTokensForExactETH	External ❗	●	NO ❗
└	swapExactTokensForETH	External ❗	●	NO ❗
└	swapETHForExactTokens	External ❗	💵	NO ❗
└	quote	External ❗		NO ❗
└	getAmountOut	External ❗		NO ❗
└	getAmountIn	External ❗		NO ❗
└	getAmountsOut	External ❗		NO ❗
└	getAmountsIn	External ❗		NO ❗
IUniswapV2Router02	Interface	IUniswapV2Router01		
└	removeLiquidityETHSupportingFeeOnTransferTokens	External ❗	●	NO ❗
└	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External ❗	●	NO ❗
└	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	●	NO ❗
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO ❗
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	●	NO ❗

Automated Analysis



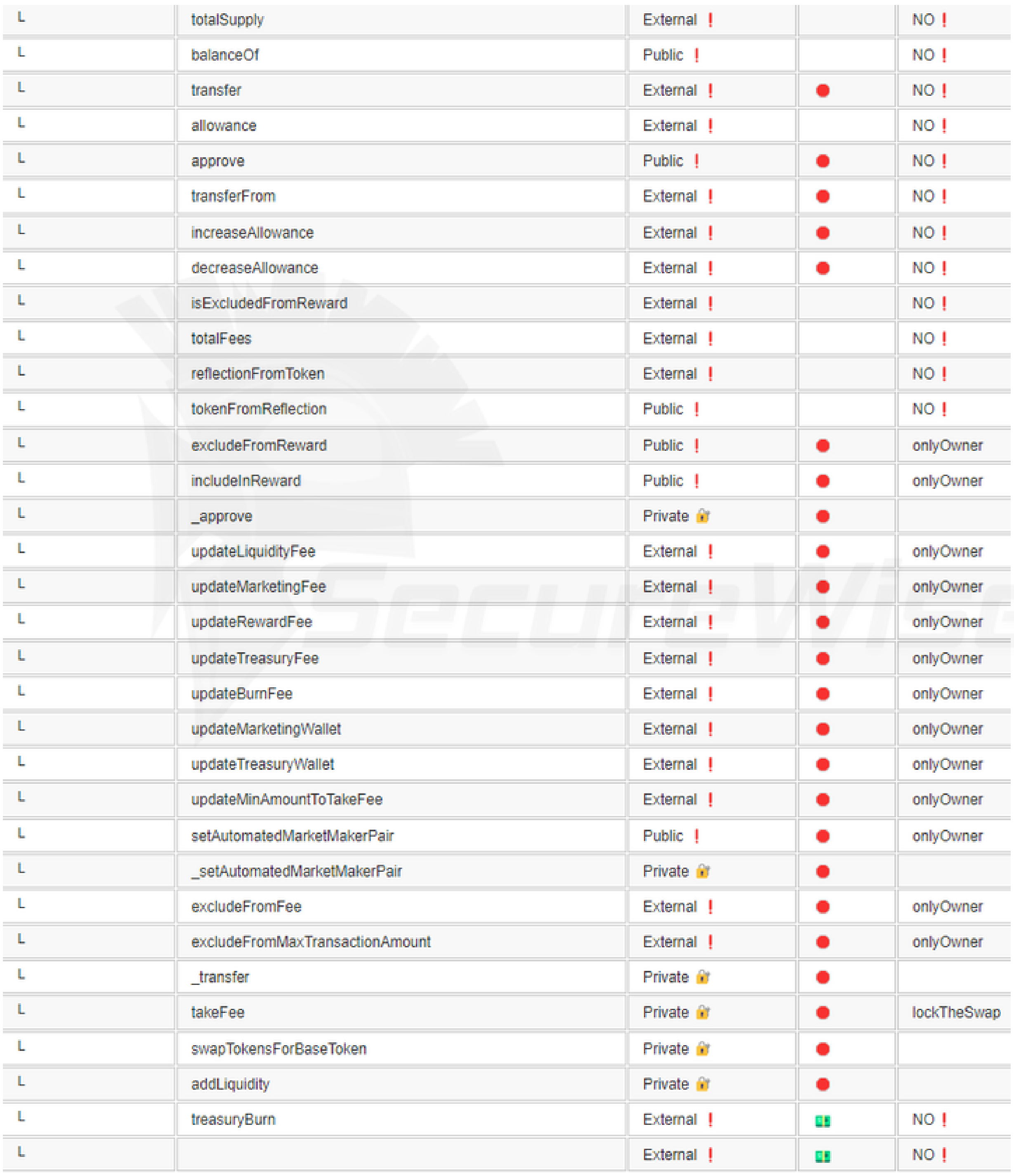
IUniswapV2Pair	Interface			
└	name	External ❗		NO ❗
└	symbol	External ❗		NO ❗
└	decimals	External ❗		NO ❗
└	totalSupply	External ❗		NO ❗
└	balanceOf	External ❗		NO ❗
└	allowance	External ❗		NO ❗
└	approve	External ❗	●	NO ❗
└	transfer	External ❗	●	NO ❗
└	transferFrom	External ❗	●	NO ❗
└	DOMAIN_SEPARATOR	External ❗		NO ❗
└	PERMIT_TYPEHASH	External ❗		NO ❗
└	nonces	External ❗		NO ❗
└	permit	External ❗	●	NO ❗
└	MINIMUM_LIQUIDITY	External ❗		NO ❗
└	factory	External ❗		NO ❗
└	token0	External ❗		NO ❗
└	token1	External ❗		NO ❗
└	getReserves	External ❗		NO ❗
└	price0CumulativeLast	External ❗		NO ❗
└	price1CumulativeLast	External ❗		NO ❗
└	kLast	External ❗		NO ❗
└	mint	External ❗	●	NO ❗
└	burn	External ❗	●	NO ❗
└	swap	External ❗	●	NO ❗
└	skim	External ❗	●	NO ❗
└	sync	External ❗	●	NO ❗
└	initialize	External ❗	●	NO ❗
IUniswapV2Factory	Interface			
└	feeTo	External ❗		NO ❗
└	feeToSetter	External ❗		NO ❗
└	getPair	External ❗		NO ❗
└	allPairs	External ❗		NO ❗

Automated Analysis



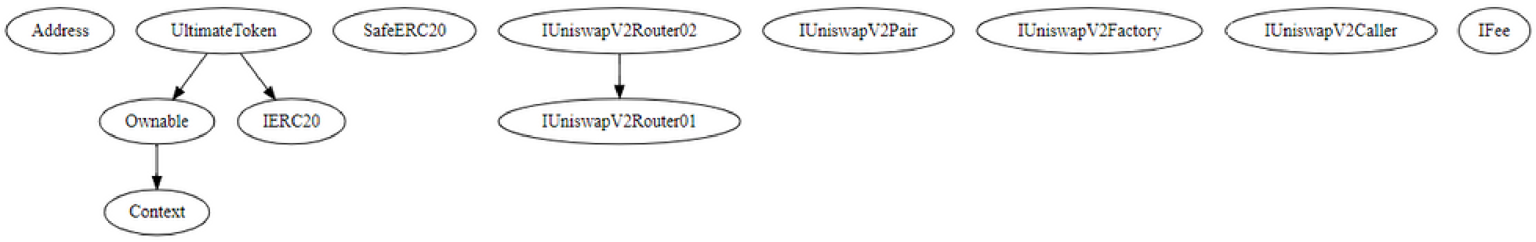
L	allPairsLength	External ❗		NO ❗
L	createPair	External ❗	●	NO ❗
L	setFeeTo	External ❗	●	NO ❗
L	setFeeToSetter	External ❗	●	NO ❗
L	INIT*CODE*PAIR_HASH	External ❗		NO ❗
IUniswapV2Caller	Interface			
L	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	●	NO ❗
IFee	Interface			
L	payFee	External ❗	💵	NO ❗
UltimateToken	Implementation	IERC20, Ownable		
L		Public ❗	💵	NO ❗
L	updateUniswapV2Pair	External ❗	●	onlyOwner
L	updateUniswapV2Router	Public ❗	●	onlyOwner
L	_tokenTransfer	Private 🔐	●	
L	updateMaxWallet	External ❗	●	onlyOwner
L	updateMaxTransactionAmount	External ❗	●	onlyOwner
L	_reflectFee	Private 🔐	●	
L	_getTValues	Private 🔐		
L	_getRate	Private 🔐		
L	_getCurrentSupply	Private 🔐		
L	removeAllFee	Private 🔐	●	
L	restoreAllFee	Private 🔐	●	
L	calculateRewardFee	Private 🔐		
L	calculateLiquidityFee	Private 🔐		
L	calculateMarketingFee	Private 🔐		
L	calculateTreasuryFee	Private 🔐		
L	calculateBurnFee	Private 🔐		
L	calculateTransferAmount	Private 🔐		
L	_takeLiquidity	Private 🔐	●	
L	name	External ❗		NO ❗
L	symbol	External ❗		NO ❗
L	decimals	External ❗		NO ❗

Automated Analysis

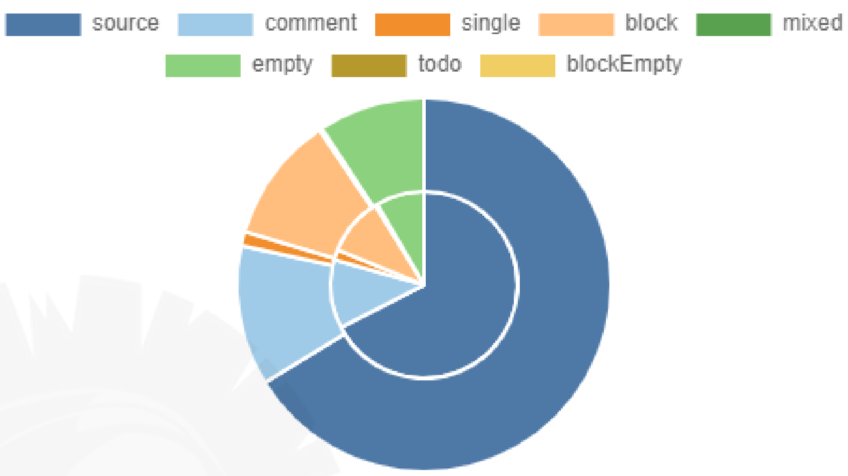


L	totalSupply	External ❗		NO ❗
L	balanceOf	Public ❗		NO ❗
L	transfer	External ❗	🔴	NO ❗
L	allowance	External ❗		NO ❗
L	approve	Public ❗	🔴	NO ❗
L	transferFrom	External ❗	🔴	NO ❗
L	increaseAllowance	External ❗	🔴	NO ❗
L	decreaseAllowance	External ❗	🔴	NO ❗
L	isExcludedFromReward	External ❗		NO ❗
L	totalFees	External ❗		NO ❗
L	reflectionFromToken	External ❗		NO ❗
L	tokenFromReflection	Public ❗		NO ❗
L	excludeFromReward	Public ❗	🔴	onlyOwner
L	includeInReward	Public ❗	🔴	onlyOwner
L	_approve	Private 🔐	🔴	
L	updateLiquidityFee	External ❗	🔴	onlyOwner
L	updateMarketingFee	External ❗	🔴	onlyOwner
L	updateRewardFee	External ❗	🔴	onlyOwner
L	updateTreasuryFee	External ❗	🔴	onlyOwner
L	updateBurnFee	External ❗	🔴	onlyOwner
L	updateMarketingWallet	External ❗	🔴	onlyOwner
L	updateTreasuryWallet	External ❗	🔴	onlyOwner
L	updateMinAmountToTakeFee	External ❗	🔴	onlyOwner
L	setAutomatedMarketMakerPair	Public ❗	🔴	onlyOwner
L	_setAutomatedMarketMakerPair	Private 🔐	🔴	
L	excludeFromFee	External ❗	🔴	onlyOwner
L	excludeFromMaxTransactionAmount	External ❗	🔴	onlyOwner
L	_transfer	Private 🔐	🔴	
L	takeFee	Private 🔐	🔴	lockTheSwap
L	swapTokensForBaseToken	Private 🔐	🔴	
L	addLiquidity	Private 🔐	🔴	
L	treasuryBurn	External ❗	💵	NO ❗
L		External ❗	💵	NO ❗

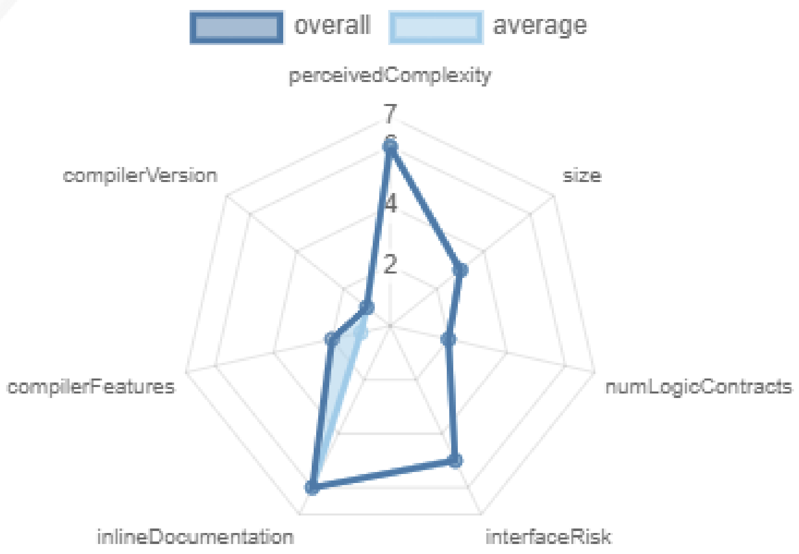
Inheritance Graph



Source Lines

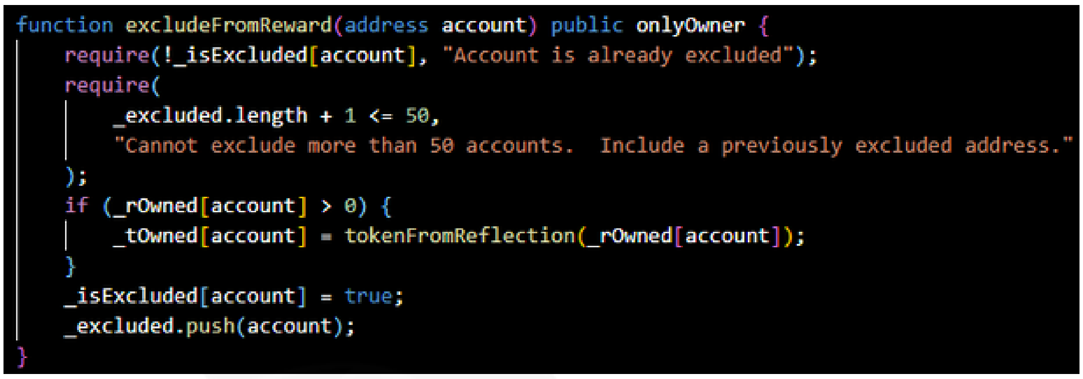


Risk



Manual Review

Owner can include/exclude addresses from rewards


```
function excludeFromReward(address account) public onlyOwner {
    require(!_isExcluded[account], "Account is already excluded");
    require(
        _excluded.length + 1 <= 50,
        "Cannot exclude more than 50 accounts.  Include a previously excluded address."
    );
    if (_rOwned[account] > 0) {
        _tOwned[account] = tokenFromReflection(_rOwned[account]);
    }
    _isExcluded[account] = true;
    _excluded.push(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 carefuly 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 change total buy/sell fee not greater than 30%

updateLiquidityFee()
updateMarketingFee()
updateRewardFee()
updateTreasuryFee()
updateBurnFee()

Description

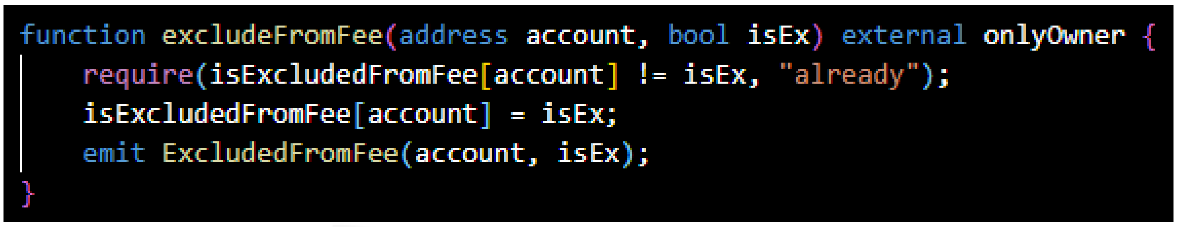
The functions updates the fees associated with selling and buying a particular token. ***Buy/Sell fees do not exceed 30% of the total***

Recommendation

it is recommended to use a proper access control mechanism. You should carefuly 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 excludeFromFee(address account, bool isEx) external onlyOwner {
    require(isExcludedFromFee[account] != isEx, "already");
    isExcludedFromFee[account] = isEx;
    emit ExcludedFromFee(account, isEx);
}
```

Description

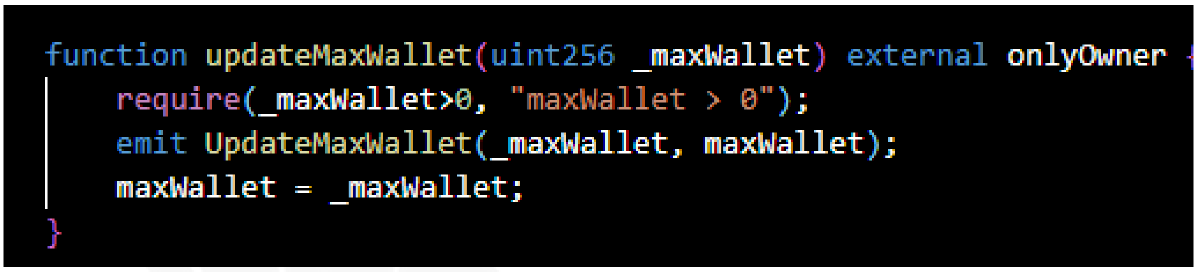
Authorizing privileged roles to exclude accounts from fees. After excluding the user from accounts, the user trades without paying a any fee and the other user sees it). But may apply in some cases like (owner wallets, contract, marketing-treasury wallets etc..)

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 update max wallet amount greater than 0



```
function updateMaxWallet(uint256 _maxWallet) external onlyOwner {
    require(_maxWallet>0, "maxWallet > 0");
    emit UpdateMaxWallet(_maxWallet, maxWallet);
    maxWallet = _maxWallet;
}
```

Description

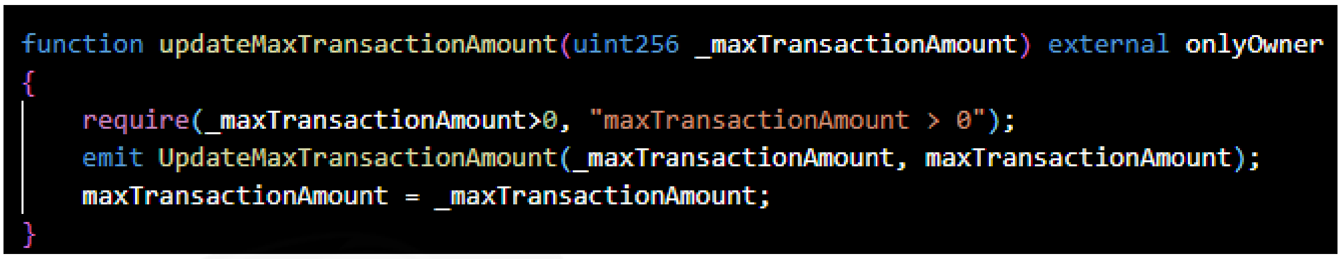
*The function about updating the maximum amount of tokens that can be held in a single wallet address. The function takes a single input parameter "**_maxWallet**", which represents the new maximum wallet amount. The function checking that the new maximum wallet amount is greater than zero.*

Recommendation

You should consider adding additional checks to ensure that the maximum wallet amount is set to a reasonable value. You should careffuly manage the private key of the owner's account.

Manual Review

Owner can update max tx amount with an amount greater than 0



```
function updateMaxTransactionAmount(uint256 _maxTransactionAmount) external onlyOwner
{
    require(_maxTransactionAmount>0, "maxTransactionAmount > 0");
    emit UpdateMaxTransactionAmount(_maxTransactionAmount, maxTransactionAmount);
    maxTransactionAmount = _maxTransactionAmount;
}
```

Description

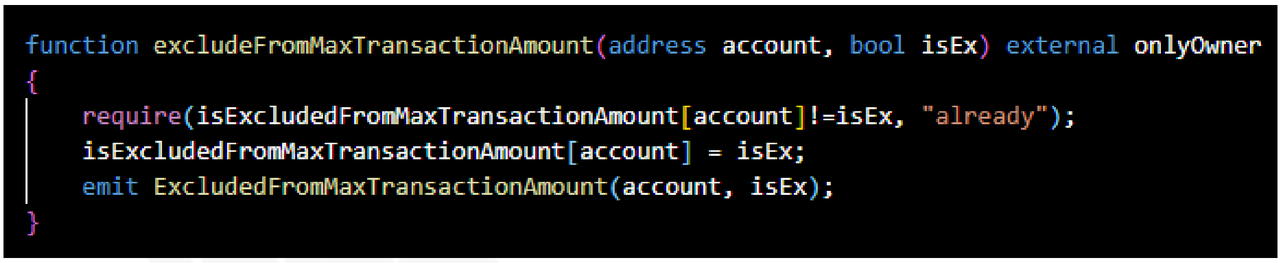
*The function about updating the maximum amount of tokens that can be transacted in a single transaction. The function takes a single input parameter "**_maxTransactionAmount**", which represents the new maximum transaction amount. The function checking that the new maximum transaction amount is greater than zero.*

Recommendation

You should consider adding additional checks to ensure that the maximum transaction amount is set to a reasonable value. You should careffuly manage the private key of the owner's account.

Manual Review

Owner can exclude/include addresses from max transaction amount



```
function excludeFromMaxTransactionAmount(address account, bool isEx) external onlyOwner
{
    require(isExcludedFromMaxTransactionAmount[account]!=isEx, "already");
    isExcludedFromMaxTransactionAmount[account] = isEx;
    emit ExcludedFromMaxTransactionAmount(account, isEx);
}
```

Description

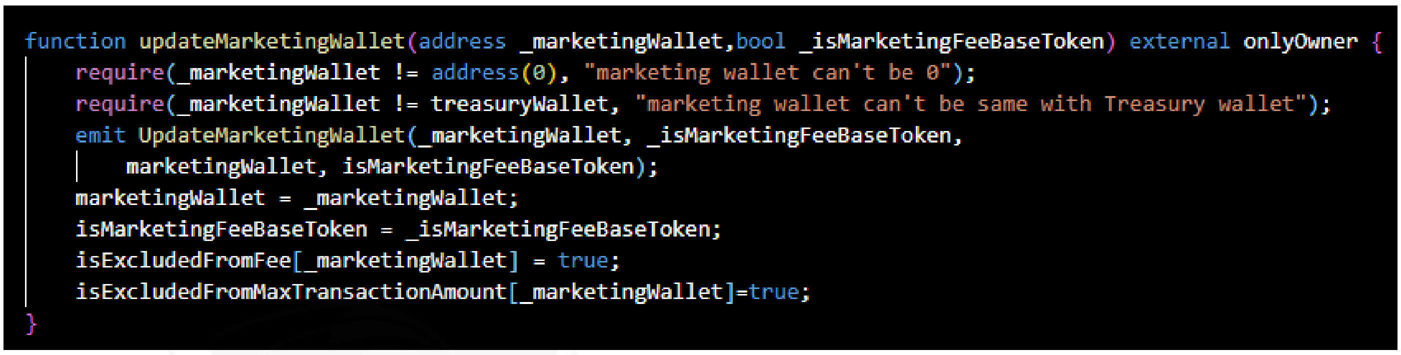
*The function allows the contract owner to exclude or include specific wallet addresses from the maximum transaction amount restriction. Function takes two input parameters, "**account**" which is the address to be excluded or included, and "**isEx**" which is a boolean that determines whether the address is excluded or included.*

Recommendation

You should consider adding additional checks to ensure that only authorized addresses can be excluded or included in the maximum transaction amount restriction

Manual Review

Owner can change marketing wallet address



```
function updateMarketingWallet(address _marketingWallet,bool _isMarketingFeeBaseToken) external onlyOwner {
    require(_marketingWallet != address(0), "marketing wallet can't be 0");
    require(_marketingWallet != treasuryWallet, "marketing wallet can't be same with Treasury wallet");
    emit UpdateMarketingWallet(_marketingWallet, _isMarketingFeeBaseToken,
        marketingWallet, isMarketingFeeBaseToken);
    marketingWallet = _marketingWallet;
    isMarketingFeeBaseToken = _isMarketingFeeBaseToken;
    isExcludedFromFee[_marketingWallet] = true;
    isExcludedFromMaxTransactionAmount[_marketingWallet]=true;
}
```

Description

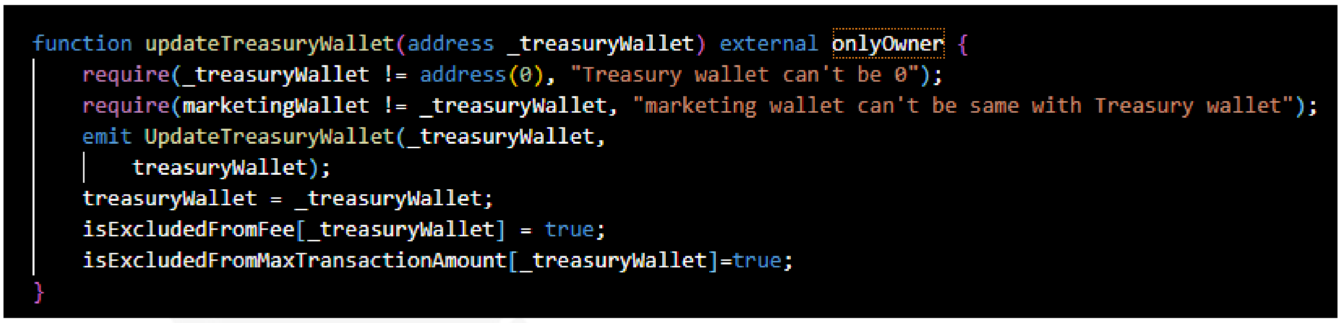
The function allows the contract owner to update the marketing wallet address and specify whether marketing fees should be charged in base tokens or etc. Function checking that the new marketing wallet address is not equal to the zero address and not equal to the treasury wallet address. Also function updates excludes the new marketing wallet address from fees and the maximum transaction amount restriction.

Recommendation

You should ensure that the new marketing wallet address is a valid and authorized address. You should carefuly 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 change treasury wallet address



```
function updateTreasuryWallet(address _treasuryWallet) external onlyOwner {
    require(_treasuryWallet != address(0), "Treasury wallet can't be 0");
    require(marketingWallet != _treasuryWallet, "marketing wallet can't be same with Treasury wallet");
    emit UpdateTreasuryWallet(_treasuryWallet,
        treasuryWallet);
    treasuryWallet = _treasuryWallet;
    isExcludedFromFee[_treasuryWallet] = true;
    isExcludedFromMaxTransactionAmount[_treasuryWallet]=true;
}
```

Description

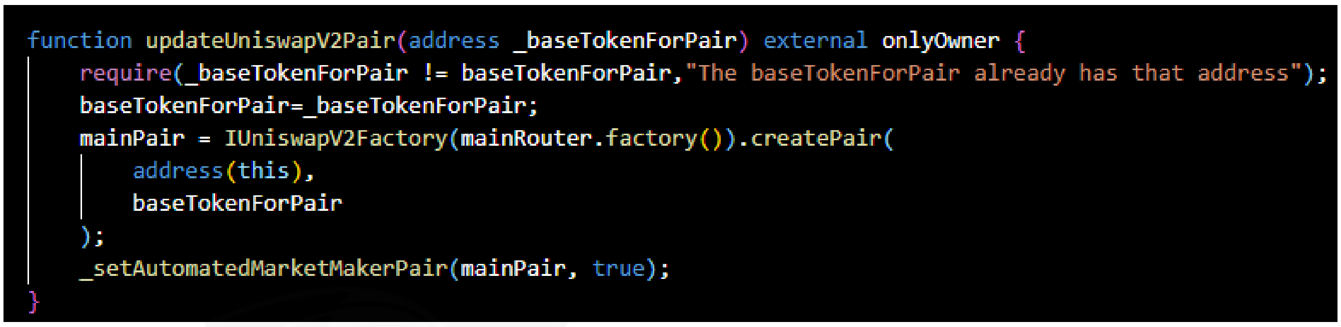
The function allows the contract owner to update the treasury wallet address and specify whether treasury fees should be charged in base tokens or etc. Function checking that the new treasury wallet address is not equal to the zero address and not equal to the marketing wallet address. Also function updates excludes the new treasury wallet address from fees and the maximum transaction amount restriction.

Recommendation

You should ensure that the new treasury wallet address is a valid and authorized address. You should careffuly 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 change UniswapV2RouterPair address



```
function updateUniswapV2Pair(address _baseTokenForPair) external onlyOwner {
    require(_baseTokenForPair != baseTokenForPair,"The baseTokenForPair already has that address");
    baseTokenForPair=_baseTokenForPair;
    mainPair = IUniswapV2Factory(mainRouter.factory()).createPair(
        address(this),
        baseTokenForPair
    );
    _setAutomatedMarketMakerPair(mainPair, true);
}
```

Description

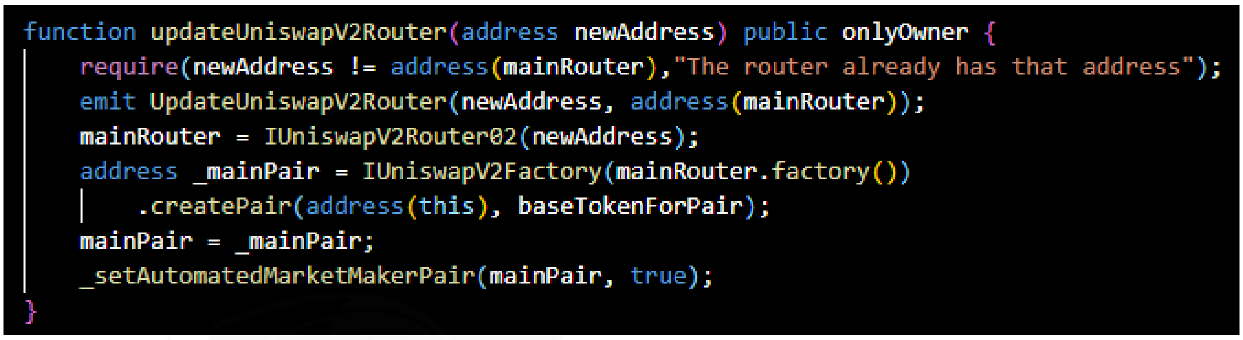
*The function allows the contract owner to update the base token for the Uniswap V2 pair. The function takes one input parameter, "**_baseTokenForPair**" which is the address of the new base token for the Uniswap V2 pair. The function checking that the new base token address is not equal to the current base token address. Updating the base token address, creates a new Uniswap V2 pair by calling the **createPair** function of the main router factory with the **contract's address** and the **new base token address** as input parameters, and **sets the automated market maker pair** to be the newly created pair.*

Recommendation

You should consider that the new base token address is a valid and authorized address, and to verify that the decision to update the base token for the Uniswap V2 pair is consistent with the project's goals and objectives

Manual Review

Owner can change UniswapV2Router address



```
function updateUniswapV2Router(address newAddress) public onlyOwner {
    require(newAddress != address(mainRouter),"The router already has that address");
    emit UpdateUniswapV2Router(newAddress, address(mainRouter));
    mainRouter = IUniswapV2Router02(newAddress);
    address _mainPair = IUniswapV2Factory(mainRouter.factory())
        .createPair(address(this), baseTokenForPair);
    mainPair = _mainPair;
    _setAutomatedMarketMakerPair(mainPair, true);
}
```

Description

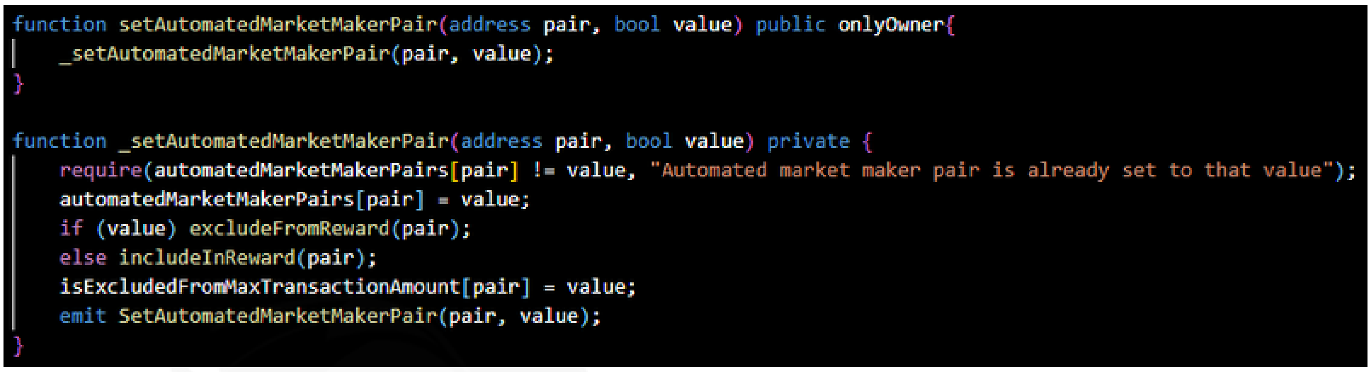
*The function allows the contract owner to update the Uniswap V2 router address. The function takes one input parameter, "**newAddress**" which is the address of the new Uniswap V2 router. The function checking that the new router address is not equal to the current router address and creating a new Uniswap V2 pair by calling the **createPair** function of the main router factory with the contract's **address** and the **base token address** as input parameters. Finally, it **sets the automated market maker pair** to be the newly created pair.*

Recommendation

You should consider that that the new router address is a valid and authorized address, and to verify that the decision to update the router address is consistent with the project's goals and objectives.

Manual Review

Owner can change automated market maker pair address and status



```
function setAutomatedMarketMakerPair(address pair, bool value) public onlyOwner{
    _setAutomatedMarketMakerPair(pair, value);
}

function _setAutomatedMarketMakerPair(address pair, bool value) private {
    require(automatedMarketMakerPairs[pair] != value, "Automated market maker pair is already set to that value");
    automatedMarketMakerPairs[pair] = value;
    if (value) excludeFromReward(pair);
    else includeInReward(pair);
    isExcludedFromMaxTransactionAmount[pair] = value;
    emit SetAutomatedMarketMakerPair(pair, value);
}
```

Description

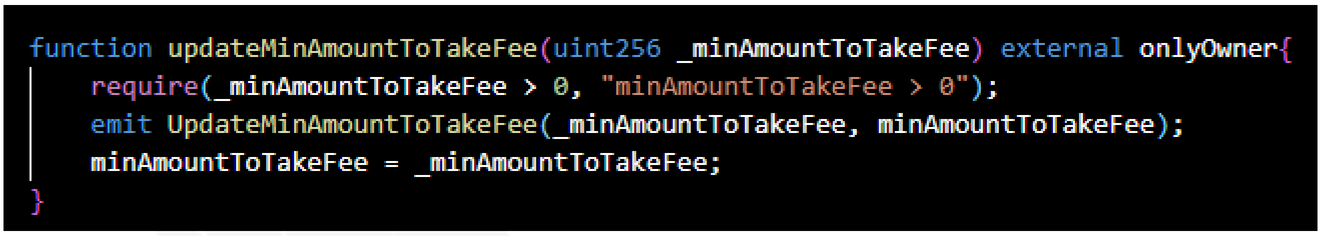
setAutomatedMarketMakerPair *function is a public function that can be called by the contract owner to set the value of* ***automatedMarketMakerPairs*** *mapping for a given pair address. The* ***_setAutomatedMarketMakerPair*** *is a private function that is called by setAutomatedMarketMakerPair to perform the actual setting of the mapping. Function is used to* ***set/unset*** *the* ***automated market maker pair*** *and* ***exclude/include*** *it from the* ***reward calculation*** *and* ***max transaction amount*** *checks accordingly.*

Recommendation

It's a necessary function for maintaining the proper functioning of the contract, and its implementation looks reasonable.

Manual Review

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



```
function updateMinAmountToTakeFee(uint256 _minAmountToTakeFee) external onlyOwner{
    require(_minAmountToTakeFee > 0, "minAmountToTakeFee > 0");
    emit UpdateMinAmountToTakeFee(_minAmountToTakeFee, minAmountToTakeFee);
    minAmountToTakeFee = _minAmountToTakeFee;
}
```

Description

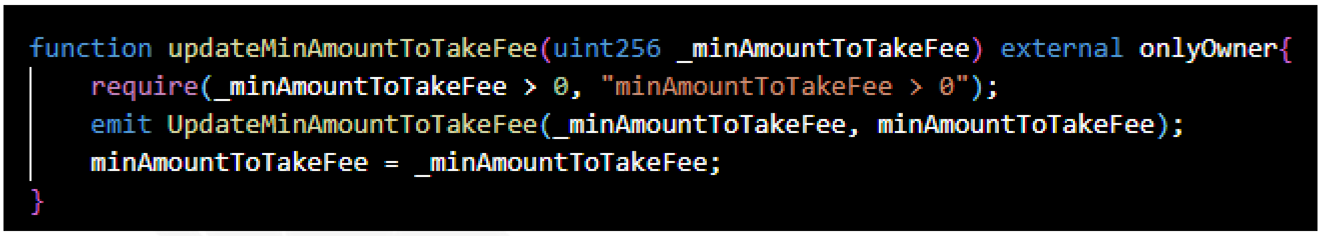
updateMinAmountToTakeFee *function updates the minimum token balance required in the contract for a transaction to take fees. The function checking contract token balance is greater then or equal to the minimum amount required to take fees (**minAmountToTakeFee**) and also function checking **_minAmountToTakeFee** > 0.*

Recommendation

minAmountToTakeFee *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 **minAmountToTakeFee** 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 **minAmountToTakeFee***

Manual Review

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



```
function updateMinAmountToTakeFee(uint256 _minAmountToTakeFee) external onlyOwner{
    require(_minAmountToTakeFee > 0, "minAmountToTakeFee > 0");
    emit UpdateMinAmountToTakeFee(_minAmountToTakeFee, minAmountToTakeFee);
    minAmountToTakeFee = _minAmountToTakeFee;
}
```

Description

updateMinAmountToTakeFee *function updates the minimum token balance required in the contract for a transaction to take fees. The function checking contract token balance is greater then or equal to the minimum amount required to take fees (**minAmountToTakeFee**) and also function checking **_minAmountToTakeFee** > 0.*

Recommendation

minAmountToTakeFee *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 **minAmountToTakeFee** 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 **minAmountToTakeFee***

SecureWise

AUDIT REPORT

SecureWise

https://securewise.info/

https://t.me/securewisehub

https://github.com/securewise