SecureWise

AUDIT REPORT

SecureWise

CORE FROG

https://securewise.info/

https://t.me/securewisehub

https://github.com/securewise

Table of 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: CoreFrog**(CoreFrog)**

Methodology: Automated Analysis, Manual Code Review

Language: Solidity

Contract Address: 0xdeF4d53903279C0FFcc7e9ab7A186de2955237C0

ContractLink: https://scan.coredao.org/address/0xdeF4d53903279C0FFcc7e9ab7A186de2955237C0

Network: Core

Supply: 10,000,000,000

Website: -

Twitter: https://twitter.com/corefrogtokens

Telegram: https://t.me/corefrog

Report Date: March 3, 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 set fees up to 100%
⚠	The owner can exclude accounts from rewards
⚠	The owner can set a bot blacklist and can use it to block any account from trading.
⚠	The owner can swap manually
⚠	The Owner can update Dead Wallet address state variables could be declared constant
⚠	The owner can exclude accounts from fees
⚠	The owner can change swap settings

Page 15 for more details

CoreFrog(CoreFrog) has succesfully **PASSED** the smart contract audit with **HIGH MEDIUM** and **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

IERC20	Interface			
└	totalSupply	External !		NO !
└	balanceOf	External !		NO !
└	transfer	External !	●	NO !
└	allowance	External !		NO !
└	approve	External !	●	NO !
└	transferFrom	External !	●	NO !
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
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 !

Automated Analysis

↳	swapExactTokensForTokensSupportingFeeOnTransferTokens	External !	●	NO !
↳	swapExactETHForTokensSupportingFeeOnTransferTokens	External !	■	NO !
↳	swapExactTokensForETHSupportingFeeOnTransferTokens	External !	●	NO !
IUniswapV2Factory	Interface			
↳	feeTo	External !		NO !
↳	feeToSetter	External !		NO !
↳	getPair	External !		NO !
↳	allPairs	External !		NO !
↳	allPairsLength	External !		NO !
↳	createPair	External !	●	NO !
↳	setFeeTo	External !	●	NO !
↳	setFeeToSetter	External !	●	NO !
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 !

Automated Analysis

L	kLast	External !		NO !
L	creation	External !	●	NO !
L	burn	External !	●	NO !
L	swap	External !	●	NO !
L	skim	External !	●	NO !
L	sync	External !	●	NO !
L	initialize	External !	●	NO !
IERC20Metadata	Interface	IERC20		
L	name	External !		NO !
L	symbol	External !		NO !
L	decimals	External !		NO !
Ownable	Implementation	Context		
L		Public !	●	NO !
L	owner	Public !		NO !
L	renounceOwnership	Public !	●	onlyOwner
L	transferOwnership	Public !	●	onlyOwner
SafeMath	Library			
L	add	Internal 🔒		
L	sub	Internal 🔒		
L	sub	Internal 🔒		
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		
L	mod	Internal 🔒		
SafeMathInt	Library			
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	sub	Internal 🔒		
L	add	Internal 🔒		
L	abs	Internal 🔒		
L	toUint256Safe	Internal 🔒		

Automated Analysis

SafeMathUint	Library			
↳	toInt256Safe	Internal 🔒		
ERC20	Implementation	Context, IERC20, IERC20Metadata		
↳		Public !	●	NO !
↳	name	Public !		NO !
↳	symbol	Public !		NO !
↳	decimals	Public !		NO !
↳	totalSupply	Public !		NO !
↳	balanceOf	Public !		NO !
↳	transfer	Public !	●	NO !
↳	allowance	Public !		NO !
↳	approve	Public !	●	NO !
↳	transferFrom	Public !	●	NO !
↳	increaseAllowance	Public !	●	NO !
↳	decreaseAllowance	Public !	●	NO !
↳	_transfer	Internal 🔒	●	
↳	_creation	Internal 🔒	●	
↳	_burn	Internal 🔒	●	
↳	_approve	Internal 🔒	●	
↳	_beforeTokenTransfer	Internal 🔒	●	
DividendPayingTokenInterface	Interface			
↳	dividendOf	External !		NO !
↳	withdrawDividend	External !	●	NO !
DividendPayingTokenOptionalInterface	Interface			
↳	withdrawableDividendOf	External !		NO !
↳	withdrawnDividendOf	External !		NO !
↳	accumulativeDividendOf	External !		NO !
DividendPayingToken	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
↳		Public !	●	ERC20
↳	distributeCAKEDividends	Public !	●	onlyOwner
↳	withdrawDividend	Public !	●	NO !
↳	_withdrawDividendOfUser	Internal 🔒	●	
↳	dividendOf	Public !		NO !
↳	withdrawableDividendOf	Public !		NO !
↳	withdrawnDividendOf	Public !		NO !

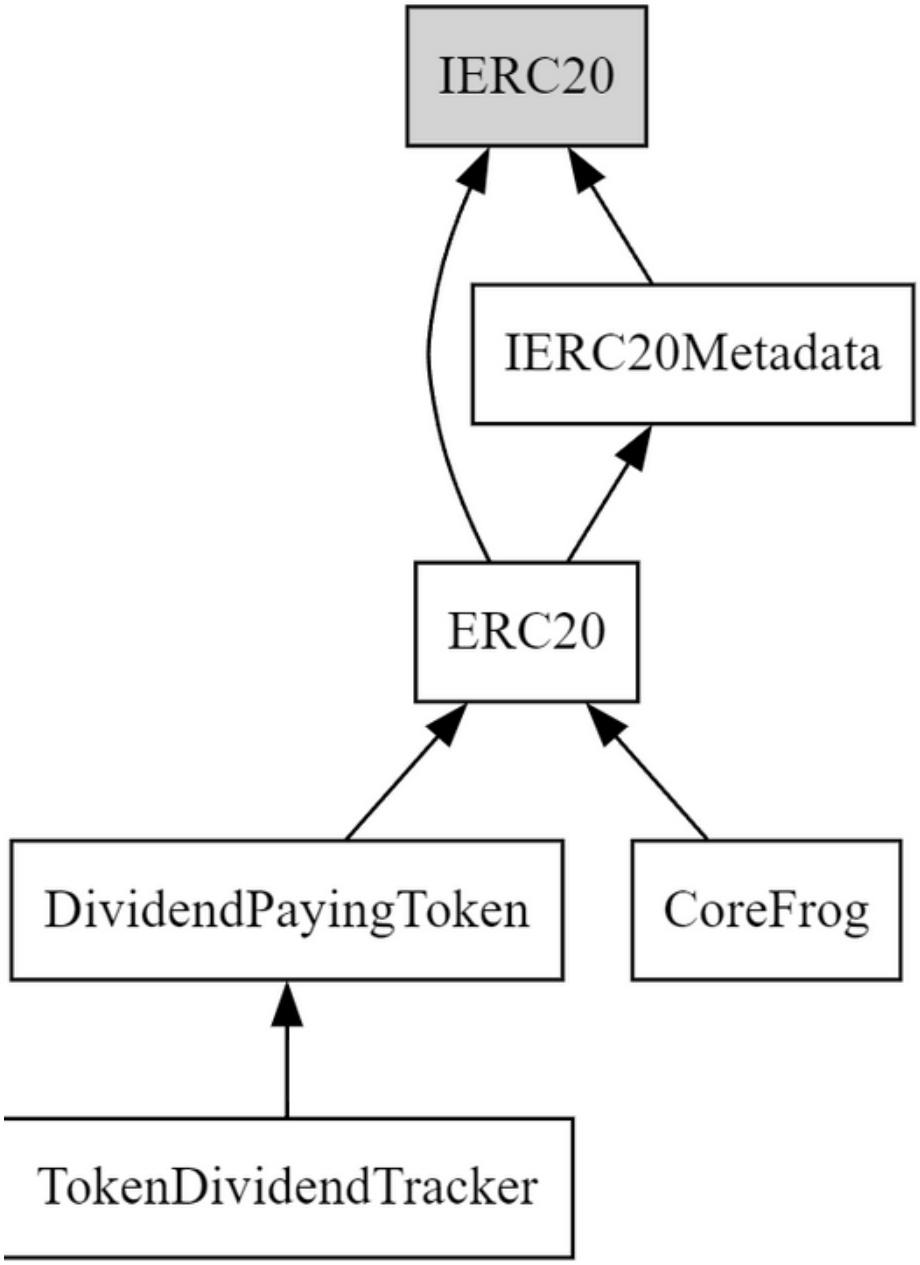
Automated Analysis

└	accumulativeDividendOf	Public !		NO !
└	_transfer	Internal 🔒	●	
└	_creation	Internal 🔒	●	
└	_burn	Internal 🔒	●	
└	_setBalance	Internal 🔒	●	
TokenDividendTracker	Implementation	Ownable, DividendPayingToken		
└		Public !	●	DividendPayingToken
└	_transfer	Internal 🔒		
└	withdrawDividend	Public !		NO !
└	setMinimumTokenBalanceForDividends	External !	●	onlyOwner
└	excludeFromDividends	External !	●	onlyOwner
└	updateClaimWait	External !	●	onlyOwner
└	getLastProcessedIndex	External !		NO !
└	getNumberOfTokenHolders	External !		NO !
└	isExcludedFromDividends	Public !		NO !
└	getAccount	Public !		NO !
└	getAccountAtIndex	Public !		NO !
└	canAutoClaim	Private 🔒		
└	setBalance	External !	●	onlyOwner
└	process	Public !	●	NO !
└	processAccount	Public !	●	onlyOwner
└	MAPGet	Public !		NO !
└	MAPGetIndexOfKey	Public !		NO !
└	MAPGetKeyAtIndex	Public !		NO !
└	MAPSize	Public !		NO !
└	MAPSet	Public !	●	NO !
└	MAPRemove	Public !	●	NO !
CoreFrog	Implementation	ERC20, Ownable		
└		Public !		ERC20
└		External !		NO !
└	updateMinimumTokenBalanceForDividends	Public !	●	onlyOwner
└	excludeFromFees	Public !	●	onlyOwner
└	excludeMultipleAccountsFromFees	Public !	●	onlyOwner
└	setMarketingWallet	External !	●	onlyOwner
└	setAutomatedMarketMakerPair	Public !	●	onlyOwner
└	EnemyAddress	External !	●	onlyOwner
└	_setAutomatedMarketMakerPair	Private 🔒	●	

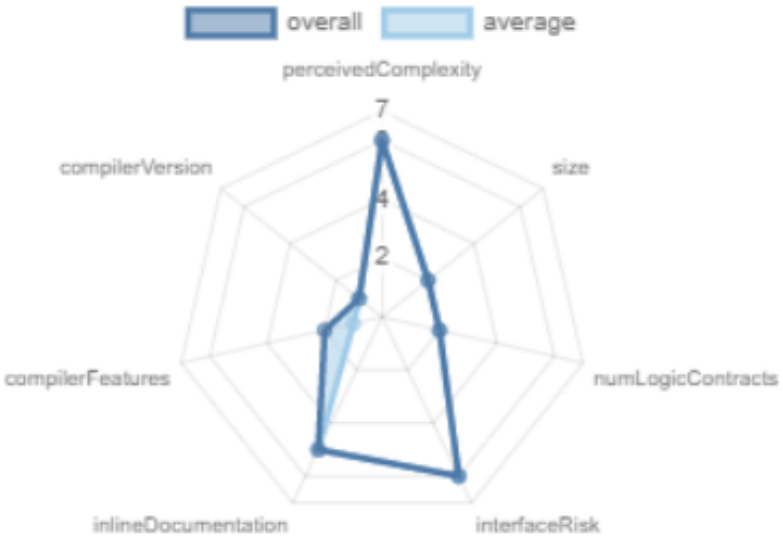
Automated Analysis

└	updateGasForProcessing	Public !	●	onlyOwner
└	updateClaimWait	External !	●	onlyOwner
└	getClaimWait	External !		NO !
└	getTotalDividendsDistributed	External !		NO !
└	isExcludedFromFees	Public !		NO !
└	withdrawableDividendOf	Public !		NO !
└	dividendTokenBalanceOf	Public !		NO !
└	excludeFromDividends	External !	●	onlyOwner
└	isExcludedFromDividends	Public !		NO !
└	getAccountDividendsInfo	External !		NO !
└	getAccountDividendsInfoAtIndex	External !		NO !
└	processDividendTracker	External !	●	NO !
└	claim	External !	●	NO !
└	getLastProcessedIndex	External !		NO !
└	getNumberOfDividendTokenHolders	External !		NO !
└	swapManual	Public !	●	onlyOwner
└	setSwapTokensAtAmount	Public !	●	onlyOwner
└	setDeadWallet	Public !	●	onlyOwner
└	setBuyLiquidityFee	Public !	●	onlyOwner
└	setSellLiquidityFee	Public !	●	onlyOwner
└	setBuyTokenRewardsFee	Public !	●	onlyOwner
└	setSellTokenRewardsFee	Public !	●	onlyOwner
└	setBuyMarketingFee	Public !	●	onlyOwner
└	setSellMarketingFee	Public !	●	onlyOwner
└	setBuyDeadFee	Public !	●	onlyOwner
└	setSellDeadFee	Public !	●	onlyOwner
└	_transfer	Internal 🔒	●	
└	swapAndSendToFee	Private 🔒	●	
└	swapAndLiquify	Private 🔒	●	
└	swapTokensForEth	Private 🔒	●	
└	swapTokensForReward	Private 🔒	●	
└	addLiquidity	Private 🔒	●	
└	swapAndSendDividends	Private 🔒	●	
└	balanceOf	Public !		NO !
└	_takeInviterFeeKt	Private 🔒	●	

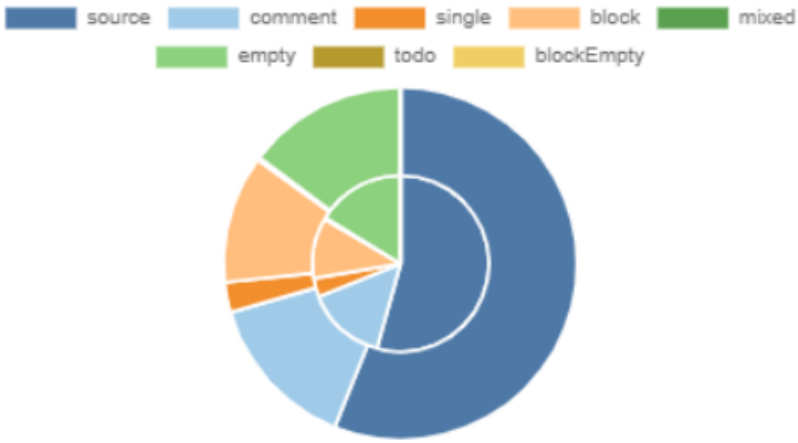
Inheritance Graph



Risk



Source Lines



Contract Summary

Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
9	8	1666	1327	850	284	851	
9	**8**	**1666**	**1327**	**850**	**284**	**851**	

Components

Contracts	Libraries	Interfaces	Abstract
5	3	8	1

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
145	6

External	Internal	Private	Pure	View
92	121	9	27	63

StateVariables

Total	Public
47	29

Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
^0.8.4		yes	—	—

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

TryCatch	Σ Unchecked
yes	—

Manual Review

The owner can set fees up to 100%

```
setBuyLiquidityFee()
setSellLiquidityFee()
setBuyTokenRewardsFee()
setSellTokenRewardsFee()
setBuyMarketingFee()
setSellMarketingFee()
setBuyDeadFee()
setSellDeadFee()
```

These functions should be provided arbitrary limits, That risk can be major problem.

Recommendation

ıt should be reasonable limits. e.g. put a require check that allows maximum limit etc.

The owner can exclude accounts from rewards

```
function excludeFromDividends(address account) external onlyOwner{
    dividendTracker.excludeFromDividends(account);
}
```

Authorizing privileged roles to exclude accounts from rewards. These cause can affect decentralization. After excluding the user from rewards the user should not get part of the community charge (and the user sees it).

Recommendation

review and fix the logic.

The owner contract tokens drain

```
function swapManual() public onlyOwner {
    uint256 contractTokenBalance = balanceOf(address(this));
    require(contractTokenBalance > 0 , "token balance zero");
    swapping = true;
    if(AmountLiquidityFee > 0) swapAndLiquify(AmountLiquidityFee);
    if(AmountTokenRewardsFee > 0) swapAndSendDividends(AmountTokenRewardsFee);
    if(AmountMarketingFee > 0) swapAndSendToFee();
    swapping = false;
}
```

The contract owner has authority to claim all the balance of the contract. The owner may take advantage of it by calling **swapManual()** functions.

Recommendation

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. That risk can be prevented by temporarily locking the contract or renouncing ownership

Manual Review

The owner can set a bot blacklist and can use it to block any account from trading.

```
function EnemyAddress(address account, bool value) external onlyOwner{
    _isEnemy[account] = value;
}
```

The contract owner has the autority to stop contracts from tx. The owner may take advantage of it by calling the blacklist/bot function.

Recommendation

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. That risk can be prevented by temporarily locking the contract or renouncing ownership

The Owner can update Dead Wallet address state variables could be declared constant

```
function setDeadWallet(address addr) public onlyOwner {
    deadWallet = addr;
}
```

State variables can be declared as constant using the constant keywords. This means that the value of the state variable cannot be changed after it has been set. If you set another wallet address for dead wallet address this is not fair logic cause it can collected all the DeadFee taxes.

Recommendation

Constant state variables can be useful when contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. The team is advised to add the constant keyword to state variables that never change.

Manual Review

The owner can exclude accounts from fees

```
function excludeFromFees(address account, bool excluded) public onlyOwner {
    if(_isExcludedFromFees[account] != excluded){
        _isExcludedFromFees[account] = excluded;
        emit ExcludeFromFees(account, excluded);
    }
}

function excludeMultipleAccountsFromFees(address[] calldata accounts, bool excluded) public onlyOwner {
    for(uint256 i = 0; i < accounts.length; i++) {
        _isExcludedFromFees[accounts[i]] = excluded;
    }

    emit ExcludeMultipleAccountsFromFees(accounts, excluded);
}
```

Authorizing privileged roles to exclude accounts from fees. These cause can affect decentralization. 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...)

Recommendation

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. That risk can be prevented by temporarily locking the contract or renouncing ownership

The owner can change swap settings

```
function setSwapTokensAtAmount(uint256 amount) public onlyOwner {
    swapTokensAtAmount = amount;
}
```

The variable **swapTokensAtAmount** sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then contract will swap a huge amount. This means that the value of price volatility.

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single tx. You should check maximum amount should be less than a fixed percantage of the total supply.

Manual Review

Unchecked return value

```
function withdrawDividend() public virtual override {
  _withdrawDividendOfUser(payable(msg.sender));
 }
```

```
processAccount(account, true); -> function setBalance(...)
```

```
function claim() external {
    dividendTracker.processAccount(payable(msg.sender), false);
  }
```

```
IERC20(rewardToken).transfer(_marketingWalletAddress,newBalance);
   -> function swapAndSendToFee(..)
```

```
uniswapV2Router.addLiquidityETH{value: ethAmount}(
      address(this),
      tokenAmount,
      0, // slippage is unavoidable
      0, // slippage is unavoidable
      address(0),
      block.timestamp
    );   -> function addLiquidity(...)
```

If the return value of a low-level call is not checked, the execution may resume even if the function call throws an error. This can lead to unexpected behaviour and break the program logic. A failed call can even be caused by an attacker, who may be able to further exploit the contract.

Recommendation

In the case that you use low-level calls, be sure to check the return value to handle possible failed calls.

Manual Review

Lacks a zero-check on set wallets function

```
DividendPayingToken.constructor(string,string,address) -
REWARD_TOKEN = _rewardTokenAddress
```

```
Ownable.constructor().msgSender - _owner = msgSender
```

```
CoreFrog.setDeadWallet(address).addr - deadWallet = addr
```

Zero-address checks as input validation on address parameters is always a best practice. This is especially true for critical addresses that are immutable and set in the constructor because they cannot be changed later. Accidentally using zero addresses here will lead to failing logic or force contract redeployment and increased gas costs.

Recommendation

Add zero-address input validation for these addresses.

Manual Review

Access Modifiers Vulnerabilities

```
updateGasForProcessing()
setSellTokenRewardsFee()
distributeCAKEDividends()
transferFrom()
setDeadWallet()
renounceOwnership()
MAPGet()
withdrawDividend()
setSellMarketingFee()
updateMinimumTokenBalanceForDividends()
decimals()
dividendOf()
decreaseAllowance()
setBuyDeadFee()
transfer()
increaseAllowance()
setBuyLiquidityFee()
swapManual()
transferOwnership()
setSellDeadFee()
process()
excludeMultipleAccountsFromFees()
setSellLiquidityFee()
setSwapTokensAtAmount()
setBuyTokenRewardsFee()
setBuyMarketingFee()
```

These functions are used as public instead of external.

Recommendation

Access control identifiers must be authenticated and set adequately to avoid possible vulnerabilities

Manual Review

Out date compiler version

```
pragma solidity ^0.8.4;
```

Compiler is set an outdated version.

Recommendation

Set and use new versions

Floating Pragma

```
pragma solidity ^0.8.4;
```

Recommendation

Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

SecureWise

AUDIT REPORT

SecureWise

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