expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

Pepe BUSD

expelee

TABLE OF CONTENTS

OVERVIEW

The Expelee team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analysed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks. According to the smart contract audit:

Audit Result	**Passed**
KYC Verification	**-**
Audit Date	**10 May 2023**

CONTRACT DETAILS

Token Name: Pepe BUSD

Symbol: PEPEBUSD

Network: Binance

Language: Solidity

Contract Address: 0x46492571242D3fEbA2Dd3A041efa0ea8185A8F30

Total Supply: 420690000000000

Contract SHA-256 Checksum:

Owner's Wallet:
0x0b09829e9d9746e4018c9aac67030700b4a7673e

Testnet:
https://testnet.bscscan.com/address/0xe04ad609ddd16a54e1560f5ef59a45169839e016

OWNER PRIVILEGES

- Owner can exclude accounts from rewards
- Owner can exclude accounts from fees
- Owner can change buy/sell and wallet to wallet transfer fees with limit up to 10%
- Trading must be enabled by the owner
- Owner can change swap settings
- Owner can change the swap tokens at amount within reasonable limit
- Owner can withdraw any token(except native token) from the contract
- Owner can change updateMinimumBalanceForDividends
- Owner can change updateClaimWait
- Owner can change updateGasForProcessing
- Owner can change updateLastProcessedIndex
- Owner can change the treasury wallet

AUDIT METHODOLOGY

Audit Details

Our comprehensive audit report provides a full overview of the audited system's architecture, smart contract codebase, and details on any vulnerabilities found within the system.

Audit Goals

The audit goal is to ensure that the project is built to protect investors and users, preventing potentially catastrophic vulnerabilities after launch , that lead to scams and rugpulls.

Code Quality

Our analysis includes both automatic tests and manual code analysis for the following aspects:

- Exploits
- Back-doors
- Vulnerability
- Accuracy
- Readability

Tools

- DE
- Open Zeppelin
- Code Analyzer
- Solidity Code
- Compiler
- Hardhat

VULNERABILITY CHECKS

Design Logic	Passed
Compiler warnings	Passed
Private user data leaks	Passed
Timestamps dependence	Passed
Integer overflow and underflow	Passed
Race conditions & reentrancy. Cross-function race conditions	Passed
Possible delays in data delivery	Passed
Oracle calls	Passed
Front Running	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Economy model	Passed
Impact of the exchange rate on the logic	Passed
Malicious event log	Passed
Scoping and declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Cross-function race conditions	Passed
Safe Zepplin module	Passed

RISK CLASSIFICATION

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and acces control issues within the code. The exploitation of these issues by malicious actors may cause serious financial damage to projects that failed to get an audit in time. We categorize these vulnerabilities by the following levels:

High Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

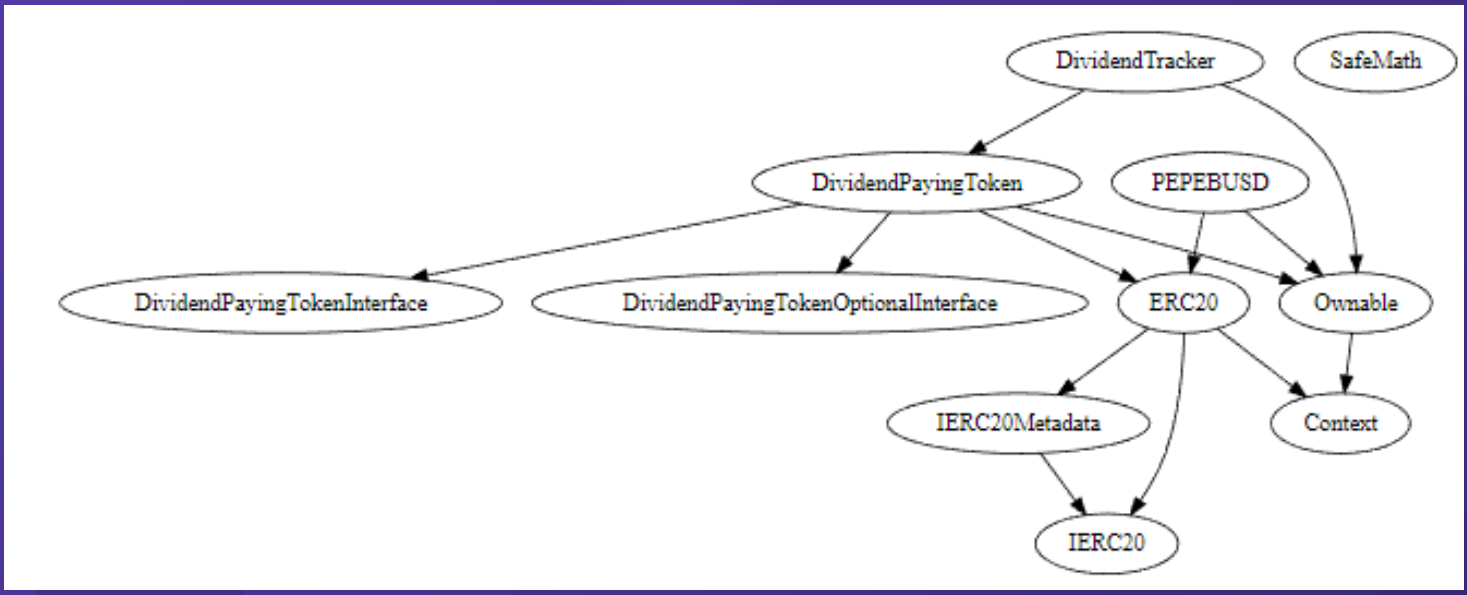
Low Risk

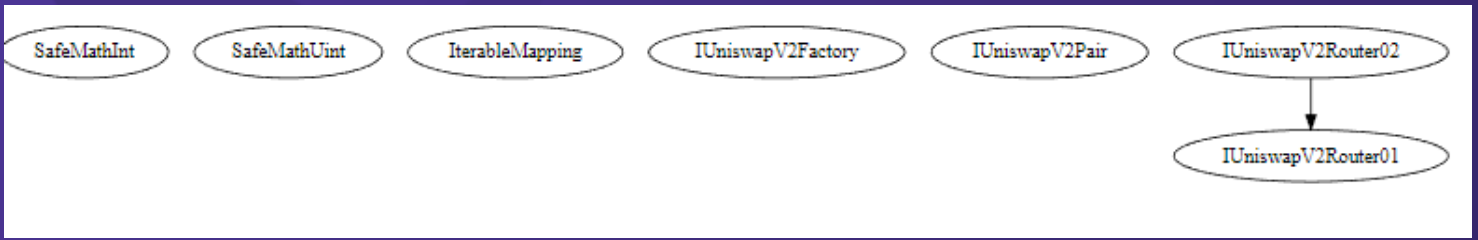
Issues on this level are minor details and warning that can remain unfixed.

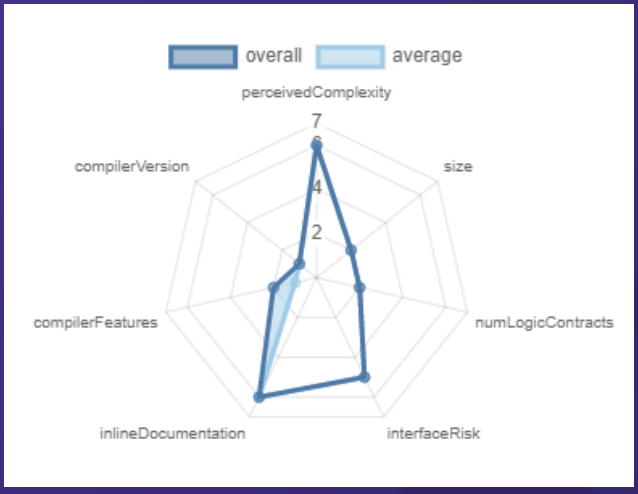
Informational

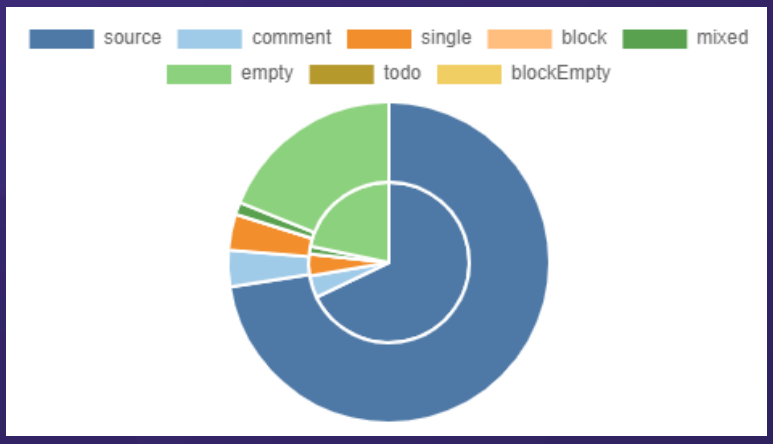
Issues on this level are minor details and warning that can remain unfixed.

INHERITANCE TREES









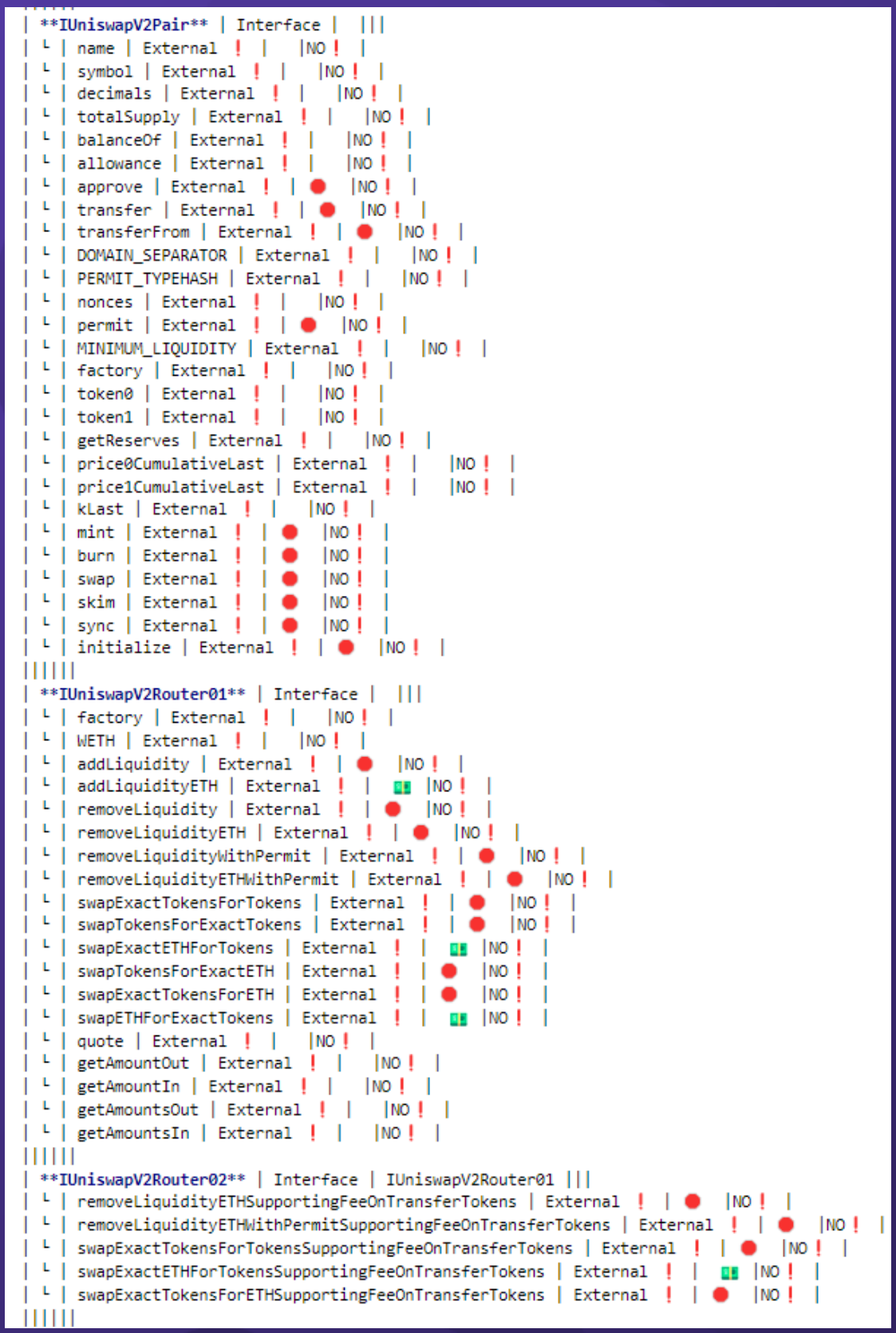
expelee

FUNCTION DETAILS

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Ownable	Implementation	Context		
└	<Constructor>	Public ❗	🛑	NO ❗
└	owner	Public ❗		NO ❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_transferOwnership	Internal 🔒	🛑	
SafeMath	Library			
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
└	mod	Internal 🔒		
SafeMathInt	Library			
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	sub	Internal 🔒		
└	add	Internal 🔒		
└	abs	Internal 🔒		
└	toUint256Safe	Internal 🔒		
SafeMathUint	Library			
└	toInt256Safe	Internal 🔒		
IterableMapping	Library			
└	get	Public ❗		NO ❗
└	getIndexOfKey	Public ❗		NO ❗
└	getKeyAtIndex	Public ❗		NO ❗
└	size	Public ❗		NO ❗
└	set	Public ❗	🛑	NO ❗
└	remove	Public ❗	🛑	NO ❗
IUniswapV2Factory	Interface			
└	feeTo	External ❗		NO ❗
└	feeToSetter	External ❗		NO ❗

expelee

FUNCTION DETAILS



expelee

FUNCTION DETAILS

IERC20	Interface			
└	totalSupply	External !		|NO !
└	balanceOf	External !		|NO !
└	allowance	External !		|NO !
└	transfer	External !	🔴	|NO !
└	approve	External !	🔴	|NO !
└	transferFrom	External !	🔴	|NO !
IERC20Metadata	Interface	IERC20		
└	name	External !		|NO !
└	symbol	External !		|NO !
└	decimals	External !		|NO !
ERC20	Implementation	Context, IERC20, IERC20Metadata		
└	<Constructor>	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 🔒	🔴		
└	_mint	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		
└	<Constructor>	Public !	🔴	ERC20
└	distributeDividends	Public !	🔴	onlyOwner
└	withdrawDividend	Public !	🔴	|NO !
└	_withdrawDividendOfUser	Internal 🔒	🔴		
└	dividendOf	Public !		|NO !
└	withdrawableDividendOf	Public !		|NO !
└	withdrawnDividendOf	Public !		|NO !
└	accumulativeDividendOf	Public !		|NO !
└	_transfer	Internal 🔒	🔴		
└	_mint	Internal 🔒	🔴		
└	_burn	Internal 🔒	🔴		
└	_setBalance	Internal 🔒	🔴		
DividendTracker	Implementation	Ownable, DividendPayingToken		
└	<Constructor>	Public !	🔴	DividendPayingToken
└	_transfer	Internal 🔒			
└	withdrawDividend	Public !		|NO !

expelee

FUNCTION DETAILS

└	updateMinimumTokenBalanceForDividends	External ❗	🔴	onlyOwner
└	excludeFromDividends	External ❗	🔴	onlyOwner
└	updateClaimWait	External ❗	🔴	onlyOwner
└	setLastProcessedIndex	External ❗	🔴	onlyOwner
└	getLastProcessedIndex	External ❗		NO ❗
└	getNumberOfTokenHolders	External ❗		NO ❗
└	getAccount	Public ❗		NO ❗
└	getAccountAtIndex	Public ❗		NO ❗
└	canAutoClaim	Private 🔐		
└	setBalance	External ❗	🔴	onlyOwner
└	process	Public ❗	🔴	NO ❗
└	processAccount	Public ❗	🔴	onlyOwner
PEPEBUSD	Implementation	ERC20, Ownable		
└	<Constructor>	Public ❗	💵	ERC20
└	<Receive Ether>	External ❗	💵	NO ❗
└	claimStuckTokens	External ❗	🔴	onlyOwner
└	isContract	Internal 🔒		
└	sendBNB	Internal 🔒	🔴	
└	_setAutomatedMarketMakerPair	Private 🔐	🔴	
└	excludeFromFees	External ❗	🔴	onlyOwner
└	isExcludedFromFees	Public ❗		NO ❗
└	updateBuyFees	External ❗	🔴	onlyOwner
└	updateSellFees	External ❗	🔴	onlyOwner
└	updateTransferFee	External ❗	🔴	onlyOwner
└	changeTreasuryWallet	External ❗	🔴	onlyOwner
└	enableTrading	External ❗	🔴	onlyOwner
└	_transfer	Internal 🔒	🔴	
└	swapAndLiquify	Private 🔐	🔴	
└	swapAndSendDividends	Private 🔐	🔴	
└	setSwapTokensAtAmount	External ❗	🔴	onlyOwner
└	setSwapEnabled	External ❗	🔴	onlyOwner
└	updateGasForProcessing	Public ❗	🔴	onlyOwner
└	updateMinimumBalanceForDividends	External ❗	🔴	onlyOwner
└	updateClaimWait	External ❗	🔴	onlyOwner
└	getClaimWait	External ❗		NO ❗
└	getTotalDividendsDistributed	External ❗		NO ❗
└	withdrawableDividendOf	Public ❗		NO ❗
└	dividendTokenBalanceOf	Public ❗		NO ❗
└	totalRewardsEarned	Public ❗		NO ❗
└	excludeFromDividends	External ❗	🔴	onlyOwner
└	getAccountDividendsInfo	External ❗		NO ❗
└	getAccountDividendsInfoAtIndex	External ❗		NO ❗
└	processDividendTracker	External ❗	🔴	NO ❗
└	claim	External ❗	🔴	NO ❗
└	claimAddress	External ❗	🔴	onlyOwner
└	getLastProcessedIndex	External ❗		NO ❗
└	setLastProcessedIndex	External ❗	🔴	onlyOwner
└	getNumberOfDividendTokenHolders	External ❗		NO ❗

MANUAL REVIEW

Severity Criteria

Expelee assesses the severity of disclosed vulnerabilities according to methodology based on OWASP standarts.

Vulnerabilities are dividend into three primary risk categroies:
High
Medium
Low

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious input handling
- Escalation of privileges
- Arithmetic
- Gas use

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

expelee

FINDINGS

Findings	Severity	Found
High Risk	High	1
Medium Risk	Medium	0
Low Risk	Low	5
Suggestion & discussion	Informational	6
Gas Optimizations	Gas Opt.	0

HIGH RISK FINDING

Trading must be enabled by the owner

Severity : High

Type : Centralisation Risk

Overview

Function enables trading by setting the **tradingEnabled** and **swapEnabled** flags to **true**

```
function enableTrading() external onlyOwner{
    require(!tradingEnabled, "Trading already enabled.");
    tradingEnabled = true;
    swapEnabled = true;
}
```

Recommendation

It is recommended to add additional access control measures, such as multi-factor authentication or time-based restrictions, to limit the number of authorized users who can call these functions. The contract owner account is well secured and only accessible by authorized parties.

LOW RISK FINDING

Owner can exclude accounts from fees

Severity : Low

Overview

Excludes/Includes an address from the collection of fees

```
function excludeFromFees(address account, bool excluded) external onlyOwner {
    require(_isExcludedFromFees[account] != excluded, "Account is already set to that state");
    _isExcludedFromFees[account] = excluded;

    emit ExcludeFromFees(account, excluded);
}
```

Recommendation

It is recommended to add additional access control measures, such as multi-factor authentication or time-based restrictions, to limit the number of authorized users who can call these functions. The contract owner account is well secured and only accessible by authorized parties.

LOW RISK FINDING

Owner can change buy/sell and wallet to wallet transfer fees with limit up to 10%

Severity : Low

Overview

Functions that allows the owner of the contract to update the buy/sell/transfer fees of the contract. These functions assumes that the input parameters are valid and do not exceed the maximum limit of 10%

```
function updateBuyFees(uint256 _liquidityFeeOnBuy, uint256 _treasuryFeeOnBuy, uint256 _rewardsFeeOnBuy) external onlyOwner {
    liquidityFeeOnBuy   = _liquidityFeeOnBuy;
    treasuryFeeOnBuy    = _treasuryFeeOnBuy;
    rewardsFeeOnBuy     = _rewardsFeeOnBuy;

    totalBuyFee = _liquidityFeeOnBuy + _treasuryFeeOnBuy + _rewardsFeeOnBuy;

    require(totalBuyFee <= 10, "Buy fee cannot be more than 10%");

    emit BuyFeesUpdated(totalBuyFee);
}
```

```
function updateSellFees(uint256 _liquidityFeeOnSell, uint256 _treasuryFeeOnSell, uint256 _rewardsFeeOnSell) external onlyOwner {
    liquidityFeeOnSell   = _liquidityFeeOnSell;
    treasuryFeeOnSell    = _treasuryFeeOnSell;
    rewardsFeeOnSell     = _rewardsFeeOnSell;

    totalSellFee = _liquidityFeeOnSell + _treasuryFeeOnSell + _rewardsFeeOnSell;

    require(totalSellFee <= 10, "Sell fee cannot be more than 10%");

    emit SellFeesUpdated(totalSellFee);
}
```

```
function updateTransferFee(uint256 _transferFee) external onlyOwner {
    transferFee = _transferFee;
    require(transferFee <= 10, "Transfer fee cannot be more than 10%"); //@audit-ok - no emit
}
```

Recommendation

It is recommended to add additional access control measures, such as multi-factor authentication or time-based restrictions, to limit the number of authorized users who can call these functions. The contract owner account is well secured and only accessible by authorized parties.

LOW RISK FINDING

Owner can change swap settings

Severity : Low

Overview

setSwapEnabled function allows the contract owner to enable or disable the automatic swapping of tokens for ETH.

```
function setSwapEnabled(bool _enabled) external onlyOwner{
    require(swapEnabled != _enabled, "swapEnabled already at this state.");
    swapEnabled = _enabled;
}
```

Recommendation

It is recommended to ensure that the contract owner account is well secured and only accessible by authorized parties.

LOW RISK FINDING

Owner can change the swap tokens at amount within reasonable limit

Severity : Low

Overview

setSwapTokensAtAmount function allows the owner to set the minimum number of tokens required to trigger an automatic swap.

```
function setSwapTokensAtAmount(uint256 newAmount) external onlyOwner{
    require(newAmount > totalSupply() / 100_000, "SwapTokensAtAmount must be greater than 0.001% of total supply");
    swapTokensAtAmount = newAmount;
}
```

Recommendation

It's important to ensure that the new **swapTokensAtAmount** value is reasonable and will not adversely affect the functioning of the token or any associated systems.

LOW RISK FINDING

Unchecked transfer

Severity : Low

Overview

The return value of an external transfer/transferFrom call is not checked

```
function withdrawDividend() public virtual override {
    _withdrawDividendOfUser(payable(msg.sender)); //@audit-issue - unchecked return value
}
```

```
    processAccount(account, true); //audit-issue unchecked return value
```

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

```
sendBNB(payable(treasuryWallet), treasuryBNB); //@audit-issue - unchecked return value
```

```
uniswapV2Router.addLiquidityETH{value: newBalance}(
    address(this),
    otherHalf,
    0, // slippage is unavoidable
    0, // slippage is unavoidable
    DEAD,
    block.timestamp
); //@audit-issue - unchecked return value
```

```
function claimAddress(address claimee) external onlyOwner {
    dividendTracker.processAccount(payable(claimee), false); //@audit-issue - unchecked return value
}
```

Recommendation

Use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

INFORMATIONAL

Owner can withdraw any token(except native token) from the contract

Severity : Informational

Overview

claimStuckTokens function allows the contract owner to recover any ERC20 tokens or BNB that were mistakenly sent to the contract's address. There arerequire statement to prevent the owner from accidentally claiming the native token.

```
//@audit-ok - Owner can withdraw any token(except native token) from the contract
0 references | Control flow graph | f9d0831a
function claimStuckTokens(address token) external onlyOwner {
    require(token != address(this), "Owner cannot claim native tokens");
    if (token == address(0x0)) {
        sendBNB(payable(msg.sender), address(this).balance); //@audit-issue - unch
        return;
    }
    IERC20 ERC20token = IERC20(token);
    uint256 balance = ERC20token.balanceOf(address(this));
    ERC20token.transfer(msg.sender, balance); //@audit-issue - unchecked return va
}
```

Recommendation

It is generally considered safe for a contract owner to claim stuck tokens, but it's important to ensure that the owner is not abusing this function to steal tokens. In this implementation, there is a require statement that ensures that the **owner cannot claim the native token** of the blockchain on which the contract is deployed.

INFORMATIONAL

Owner can exclude accounts from rewards

Severity : Informational

Overview

Function that allows the owner of the contract to exclude an address from receiving dividends

```
function excludeFromDividends(address account) external onlyOwner {
    require(!excludedFromDividends[account]);
    excludedFromDividends[account] = true;

    _setBalance(account, 0);
    tokenHoldersMap.remove(account);

    emit ExcludeFromDividends(account);
}
```

Recommendation

It is recommended to add additional access control measures, such as multi-factor authentication or time-based restrictions, to limit the number of authorized users who can call these functions. The contract owner account is well secured and only accessible by authorized parties.

INFORMATIONAL

Missing events

Severity : Informational

Overview

Detect missing events for critical access control parameters

```
function enableTrading() external onlyOwner{
    require(!tradingEnabled, "Trading already enabled.");
    tradingEnabled = true;
    swapEnabled = true;
}
```

```
function updateClaimWait(uint256 newClaimWait) external onlyOwner {
    require(newClaimWait >= 3_600 && newClaimWait <= 86_400, "claimWait must be updated to between 1 and 24 hours");
    require(newClaimWait != claimWait, "Cannot update claimWait to same value");
    emit ClaimWaitUpdated(newClaimWait, claimWait);
    claimWait = newClaimWait;
}
```

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

```
function updateTransferFee(uint256 _transferFee) external onlyOwner {
    transferFee = _transferFee;
    require(transferFee <= 10, "Transfer fee cannot be more than 10%"); //@audit-issue - no emit
}
```

```
function updateMinimumTokenBalanceForDividends(uint256 _newMinimumBalance) external onlyOwner {
    require(_newMinimumBalance != minimumTokenBalanceForDividends, "New mimimum balance for dividend cannot be same as current minimum balance");
    minimumTokenBalanceForDividends = _newMinimumBalance;
}
```

```
function setSwapEnabled(bool _enabled) external onlyOwner{
    require(swapEnabled != _enabled, "swapEnabled already at this state.");
    swapEnabled = _enabled;
}
```

```
function setSwapTokensAtAmount(uint256 newAmount) external onlyOwner{
    require(newAmount > totalSupply() / 100_000, "SwapTokensAtAmount must be greater than 0.001% of total supply");
    swapTokensAtAmount = newAmount;
}
```

Recommendation

Emit an event for critical parameter changes.

INFORMATIONAL

tx.origin detected

Severity : Informational

Overview

tx.origin-based protection can be abused by a malicious contract if a legitimate user interacts with the malicious contract

```
function processDividendTracker(uint256 gas) external {
    (uint256 iterations, uint256 claims, uint256 lastProcessedIndex) = dividendTracker.process(gas);
    emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, false, gas, tx.origin);
} //@audit-issue - tx.origin used in an unsafe manner
```

```
emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, true, gas, tx.origin);
```

Recommendation

It is recommended to replace the **tx.origin** variable with **msg.sender**, which refers to the address of the caller of the current function. Using **msg.sender** is a safer approach as it cannot be manipulated by an attacker.

INFORMATIONAL

Unused events

Severity : Informational

Overview
Detected unused events

```
event TransferFeesUpdated(uint256 fee1, uint256 fee2); //@audit-issue unused event
```

```
event UpdateUniswapV2Router(address indexed newAddress, address indexed oldAddress); //@audit-issue unused event
```

```
event UpdateDividendTracker(address indexed newAddress, address indexed oldAddress); //@audit-issue unused event
```

Recommendation

If an event is truly unused and serves no purpose, it is recommended to simply remove it from the code.

INFORMATIONAL

Old version of Solidity compiler

Severity : Informational

Overview

solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex pragma statement.

```
pragma solidity 0.8.17;
//@audit-issue - outdated version of solidity
```

Recommendation

Consider using the latest version of Solidity.

ABOUT EXPELEE

Expelee is a product-based aspirational Web3 start-up. Coping up with numerous solutions for blockchain security and constructing a Web3 ecosystem from deal making platform to developer hosting open platform, while also developing our own commercial and sustainable blockchain.

www.expelee.com

expeleeofficial
expelee
Expelee
expelee
expelee_official
expelee-co

expelee

Building the Futuristic Blockchain Ecosystem

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantess against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always do your own research and project yourselves from being scammed. The Expelee team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools.

Under no circumstances did Expelee receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Alway do your own research and protect yourselves from scams.

This document should not be presented as a reason to buy or not buy any particular token. The Expelee team disclaims any liability for the resulting losses.