expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

PUPPY INU

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:

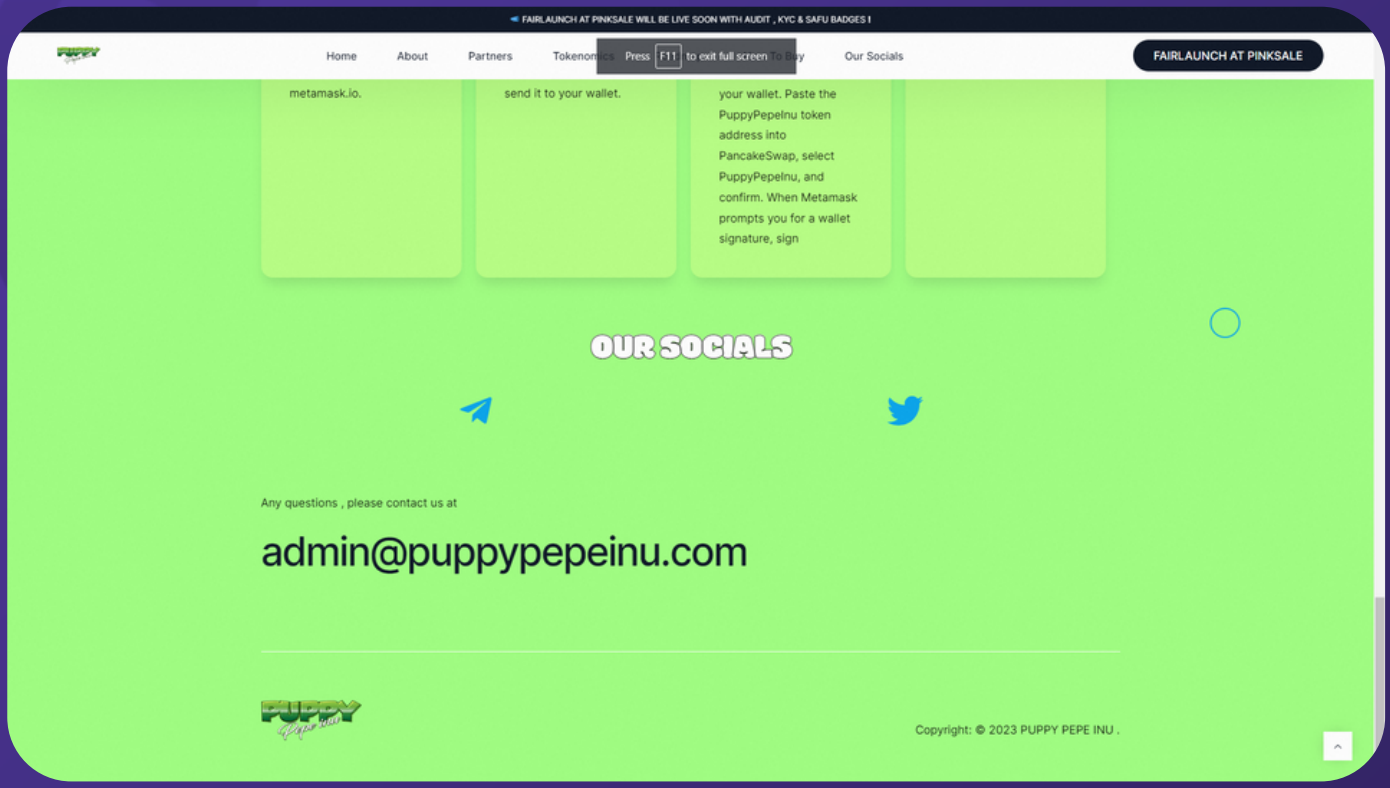
Audit Result	**Passed**
KYC Verification	**Done**
Audit Date	**19 May 2023**

PROJECT DESCRIPTION

PUPPY PEPE INU IS HERE FOR WHOM MISSED THE PEPE COIN RIDE , IT'S GOING TO MAKE HISTORY PUPPY PEPE INU IS DESIGNED IN SUCH A WAY TO KEEP THE CHART PUMPED WITH AUTO BUY BACK BURN & MARKETING !

SOCIAL MEDIA PROFILES

PUPPY INU



https://t.me/puppypepeinu

https://twitter.com/puppypepeinu

https://puppypepeinu.com/

It's always good to check the social profiles of the project, before making your investment.

Team Expelee

CONTRACT DETAILS

Token Name: Puppy Pepe Inu

Symbol: PepeInu

Network: Binance Smart Chain

Language: Solidity

Contract Address: 0x1FBF8D0d7211dB4dB3235F100D836FAd1Fca4534

Total Supply: 1000000000000

Owner's Wallet: 0x0199d9dca23b31cf17d2620b53f2bb406d8840e8

Deployer's Wallet: 0x0199d9dca23b31cf17d2620b53f2bb406d8840e8

Testnet:
https://testnet.bscscan.com/address/0x96Bd7e870Ce50692B40576f63e2E1Bd1212D53B3

OWNER PRIVILEGES

- Owner can exclude account from fees
- Owner can change fees max 10%
- Trading must be enabled by the owner
- Owner can change max wallet token amount within reasonable limits
- Owner can change swap settings
- Owner can change buyback status settings
- Owner can change buybackAmount and buybackThreshold settings
- Owner can withdraw claim stuck bnb and and tokens except native token before adding liquidity
- Owner can exclude account from protection implemantation
- Owner can change protection settings
- Owner can update router address before adding liquidity
- Owner can add new LP pair

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.

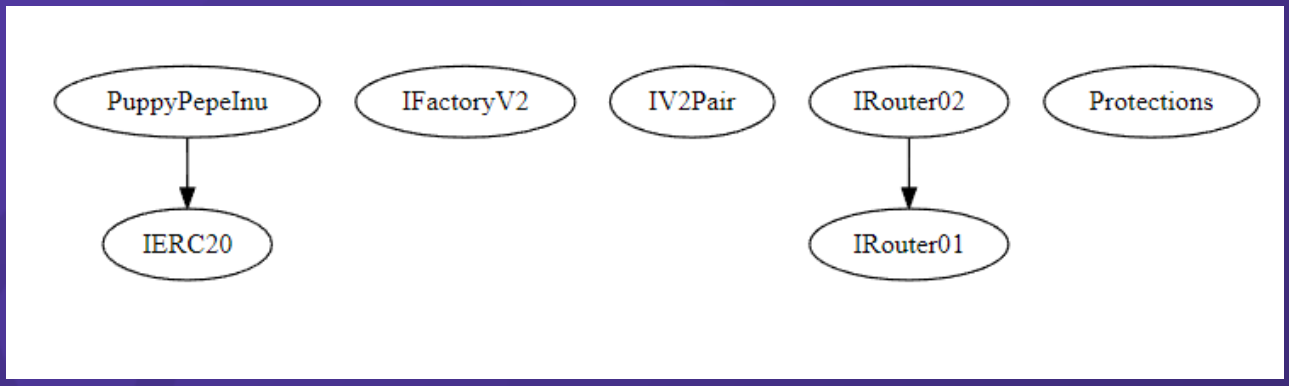
Low Risk

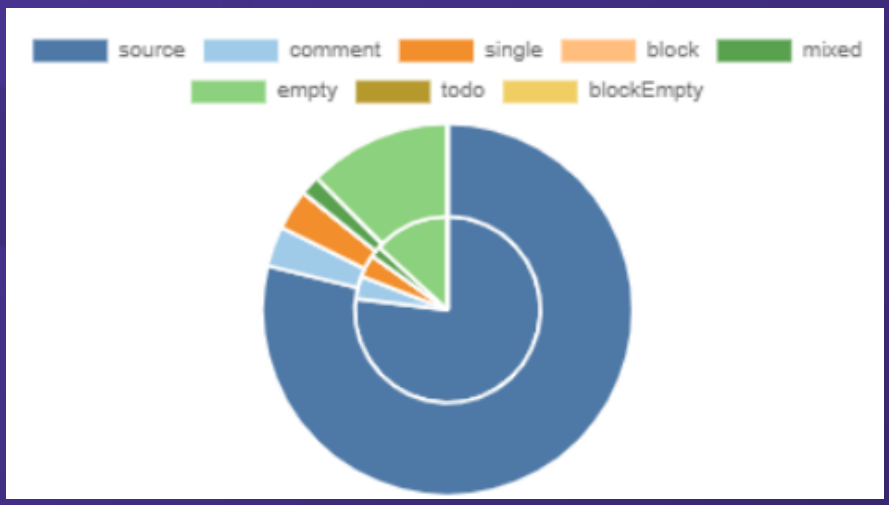
Issues on this level are minor details and warning that can remain unfixed.

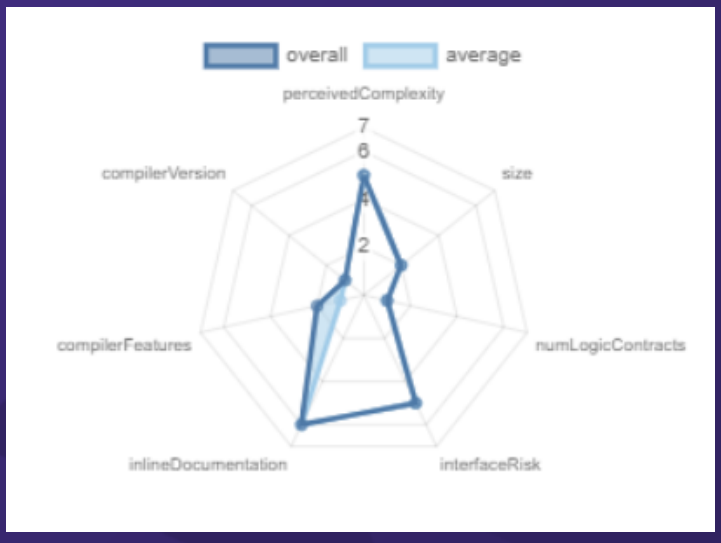
Informational

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

INHERITANCE TREES







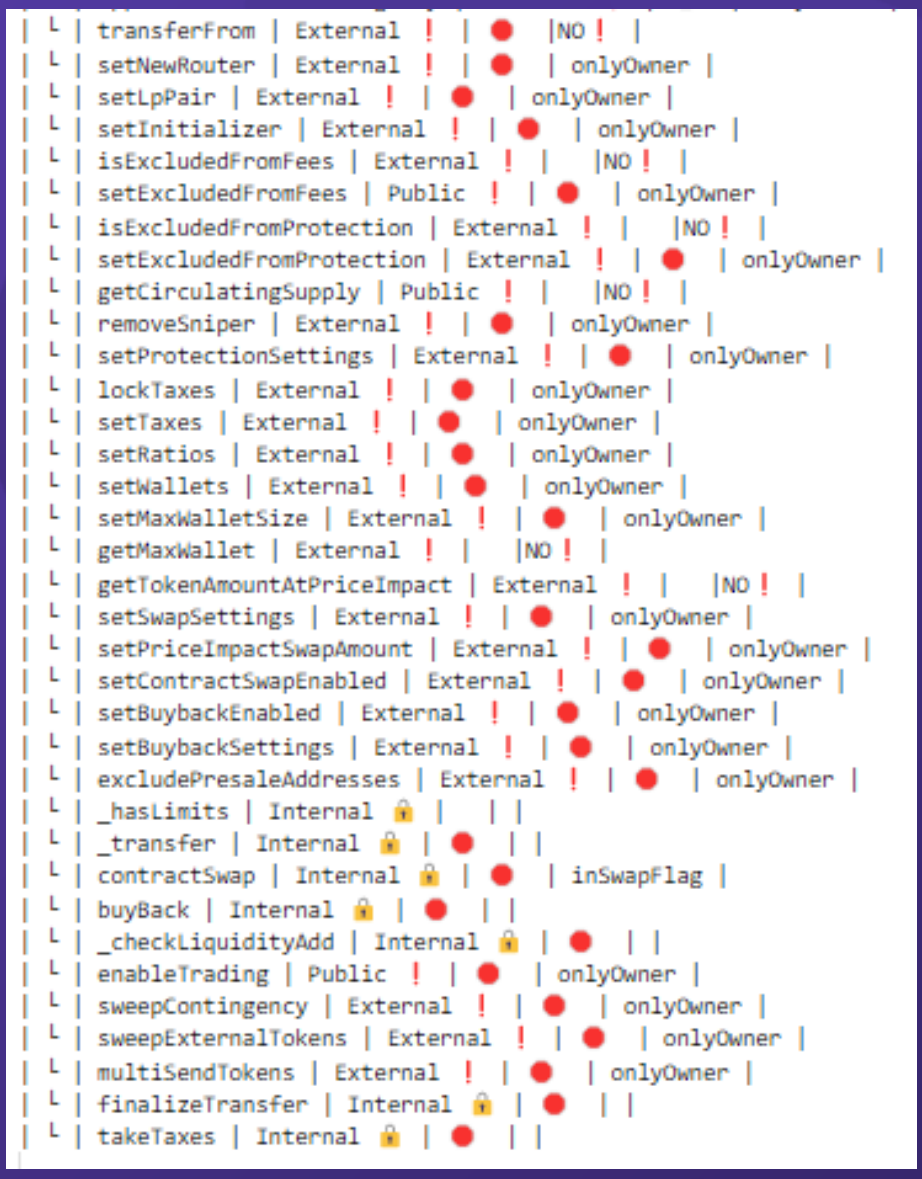
expelee

FUNCTION DETAILS

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
└	totalSupply	External ❗		NO ❗
└	decimals	External ❗		NO ❗
└	symbol	External ❗		NO ❗
└	name	External ❗		NO ❗
└	getOwner	External ❗		NO ❗
└	balanceOf	External ❗		NO ❗
└	transfer	External ❗	🔴	NO ❗
└	allowance	External ❗		NO ❗
└	approve	External ❗	🔴	NO ❗
└	transferFrom	External ❗	🔴	NO ❗
IFactoryV2	Interface			
└	getPair	External ❗		NO ❗
└	createPair	External ❗	🔴	NO ❗
IV2Pair	Interface			
└	factory	External ❗		NO ❗
└	getReserves	External ❗		NO ❗
└	sync	External ❗	🔴	NO ❗
IRouter01	Interface			
└	factory	External ❗		NO ❗
└	WETH	External ❗		NO ❗
└	addLiquidityETH	External ❗	💸	NO ❗
└	addLiquidity	External ❗	🔴	NO ❗
└	swapExactETHForTokens	External ❗	💸	NO ❗
└	getAmountsOut	External ❗		NO ❗
└	getAmountsIn	External ❗		NO ❗
IRouter02	Interface	IRouter01		
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🔴	NO ❗
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💸	NO ❗
└	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🔴	NO ❗
└	swapExactTokensForTokens	External ❗	🔴	NO ❗
Protections	Interface			
└	checkUser	External ❗	🔴	NO ❗
└	setLaunch	External ❗	🔴	NO ❗
└	getInits	External ❗	🔴	NO ❗
└	setLpPair	External ❗	🔴	NO ❗
└	setProtections	External ❗	🔴	NO ❗
└	removeSniper	External ❗	🔴	NO ❗
PuppyPepeInu	Implementation	IERC20		
└	<Constructor>	Public ❗	💸	NO ❗
└	transferOwner	External ❗	🔴	onlyOwner
└	renounceOwnership	External ❗	🔴	onlyOwner
└	setOperator	Public ❗	🔴	NO ❗
└	renounceOriginalDeployer	External ❗	🔴	NO ❗
└	<Receive Ether>	External ❗	💸	NO ❗
└	totalSupply	External ❗		NO ❗
└	decimals	External ❗		NO ❗
└	symbol	External ❗		NO ❗
└	name	External ❗		NO ❗
└	getOwner	External ❗		NO ❗
└	allowance	External ❗		NO ❗
└	balanceOf	Public ❗		NO ❗
└	transfer	Public ❗	🔴	NO ❗
└	approve	External ❗	🔴	NO ❗
└	_approve	Internal 🔒	🔴	
└	approveContractContingency	External ❗	🔴	onlyOwner

expelee

FUNCTION DETAILS



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	0
Medium Risk	Medium	0
Low Risk	Low	7
Suggestion & discussion	Informational	3
Gas Optimizations	Gas Opt.	0

LOW RISK FINDING

Owner can exclude accounts from fees

Severity : Low

Overview

Excludes/Includes an address from the collection of fees

```
function setExcludedFromFees(address account, bool enabled) public onlyOwner {
    _isExcludedFromFees[account] = enabled;
}
```

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 fee max 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 setTaxes(uint16 buyFee, uint16 sellFee, uint16 transferFee) external onlyOwner {
    require(!taxesAreLocked, "Taxes are locked.");
    require(buyFee <= maxBuyTaxes
            && sellFee <= maxSellTaxes
            && transferFee <= maxTransferTaxes,
            "Cannot exceed maximums.");
    _taxRates.buyFee = buyFee;
    _taxRates.sellFee = sellFee;
    _taxRates.transferFee = transferFee;
}
```

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

Trading must be enabled by the owner

Severity : Low

Overview

Function enables trading by setting the **tradingEnabled** true

```
function enableTrading() public onlyOwner {
    require(!tradingEnabled, "Trading already enabled!");
    require(_hasLiqBeenAdded, "Liquidity must be added.");
    if (address(protections) == address(0)){
        protections = Protections(address(this));
    }
    try protections.setLaunch(lpPair, uint32(block.number), uint64(block.timestamp), _decimals) {} catch {}
    try protections.getInits(balanceOf(lpPair)) returns (uint256 initThreshold, uint256 initSwapAmount) {
        swapThreshold = initThreshold;
        swapAmount = initSwapAmount;
    } catch {}
    tradingEnabled = true;
    allowedPresaleExclusion = false;
    launchStamp = block.timestamp;
}
```

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 max wallet token amount within reasonable limits

Severity : Low

Overview

setMaxWalletSize function that allows the contract owner to set the maximum wallet size as a percentage of the total token supply.

```
function setMaxWalletSize(uint256 percent, uint256 divisor) external onlyOwner {
    require((_tTotal * percent) / divisor >= (_tTotal / 100), "Max Wallet amt must be above 1% of total supply.");
    _maxWalletSize = (_tTotal * percent) / divisor;
}
```

Recommendation

Verify that appropriate access control mechanisms are in place to restrict the **setMaxWalletSize** function to the contract owner only. Ensure that the **onlyOwner** modifier is correctly implemented and that ownership cannot be easily transferred or compromised.

LOW RISK FINDING

Owner can change swap setting

Severity : Low

Overview

Functions allows the contract owner to enable or disable the automatic swapping. and setting swapThreshold, swapAmount.

```
function setSwapSettings(uint256 thresholdPercent, uint256 thresholdDivisor, uint256 amountPercent, uint256 amountDivisor) external onlyOwner {
    swapThreshold = (_tTotal * thresholdPercent) / thresholdDivisor;
    swapAmount = (_tTotal * amountPercent) / amountDivisor;
    require(swapThreshold <= swapAmount, "Threshold cannot be above amount.");
    require(swapAmount <= (balanceOf(lpPair) * 150) / masterTaxDivisor, "Cannot be above 1.5% of current PI.");
    require(swapAmount >= _tTotal / 1_000_000, "Cannot be lower than 0.00001% of total supply.");
    require(swapThreshold >= _tTotal / 1_000_000, "Cannot be lower than 0.00001% of total supply.");
}
```

```
function setContractSwapEnabled(bool swapEnabled, bool priceImpactSwapEnabled) external onlyOwner {
    contractSwapEnabled = swapEnabled;
    piContractSwapsEnabled = priceImpactSwapEnabled;
    emit ContractSwapEnabledUpdated(swapEnabled);
}
```

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 buyback status settings

Severity : Low

Overview

buybackEnabled that determines whether a buyback mechanism is enabled or not. The contract owner can use the function **setBuybackEnabled** to enable or disable the buyback mechanism.

```
function setBuybackEnabled(bool enabled) external onlyOwner {
    buybackEnabled = 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 buybackAmount and buybackThreshold settings

Severity : Low

Overview

setBuybackSettings function that allows the contract owner to set the buyback settings. The function takes four parameters: threshold, thresholdMultiplier, amount, and amountMultiplier. The values are used to calculate the buybackThreshold and buybackAmount variables by multiplying the respective values with 10 raised to the power of the corresponding multiplier.

```
function setBuybackSettings(uint256 threshold, uint256 thresholdMultiplier, uint256 amount, uint256 amountMultiplier) external onlyOwner {
    buybackThreshold = threshold * 10**thresholdMultiplier;
    buybackAmount = amount * 10**amountMultiplier;
}
```

Recommendation

Verify that appropriate access control mechanisms are in place to restrict the **setBuybackSettings** function to the contract owner only. Ensure that the onlyOwner modifier is correctly implemented and that ownership cannot be easily transferred or compromised.

LOW RISK FINDING

Owner can withdraw claim stuck bnb and and tokens except native token before adding liquidity

Severity : Informational

Overview

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

```
function sweepContingency() external onlyOwner {
    require(!_hasLiqBeenAdded, "Cannot call after liquidity.");
    payable(_owner).transfer(address(this).balance);
}


function sweepExternalTokens(address token) external onlyOwner {
    if (_hasLiqBeenAdded) {
        require(token != address(this), "Cannot sweep native tokens.");
    }
    IERC20 TOKEN = IERC20(token);
    TOKEN.transfer(_owner, TOKEN.balanceOf(address(this)));
}
```

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 FINDING

Owner can update router address before adding liquidity

Severity : Informational

Overview

setNewRouter that allows the contract owner to change the router address used for swapping tokens. If liquidity has been added, the function throws an exception and does not allow changing the router

```
function setNewRouter(address newRouter) external onlyOwner {
    require(!_hasLiqBeenAdded, "Cannot change after liquidity.");
    IRouter02 _newRouter = IRouter02(newRouter);
    address get_pair = IFactoryV2(_newRouter.factory()).getPair(address(this), _newRouter.WETH());
    lpPairs[lpPair] = false;
    if (get_pair == address(0)) {
        lpPair = IFactoryV2(_newRouter.factory()).createPair(address(this), _newRouter.WETH());
    }
    else {
        lpPair = get_pair;
    }
    dexRouter = _newRouter;
    lpPairs[lpPair] = true;
    _approve(address(this), address(dexRouter), type(uint256).max);
}
```

Recommendation

Implement appropriate validation checks on the **newRouter** address to prevent potential vulnerabilities, such as mistakenly setting an incorrect or malicious address. Verify that appropriate access control mechanisms are in place to restrict the **setNewRouter** function to the contract owner only.

INFORMATIONAL FINDING

Owner can add new LP pair.

Severity : Informational

Overview

setLpPair that allows the contract owner to enable or disable an LP (liquidity pool) pair.

```
function setLpPair(address pair, bool enabled) external onlyOwner {
    if (!enabled) {
        lpPairs[pair] = false;
        protections.setLpPair(pair, false);
    } else {
        if (timeSinceLastPair != 0) {
            require(block.timestamp - timeSinceLastPair > 3 days, "3 Day cooldown.");
        }
        require(!lpPairs[pair], "Pair already added to list.");
        lpPairs[pair] = true;
        timeSinceLastPair = block.timestamp;
        protections.setLpPair(pair, true);
    }
}
```

Recommendation

Verify that the **protections** contract used in the **setLpPair** function is a trusted and secure contract. Verify that appropriate access control mechanisms are in place to restrict the setLpPair function to the contract owner only. Evaluate the necessity of the cooldown period and the specific duration of 3 days. Assess whether this period aligns with the project's requirements and consider the potential impact of changing the cooldown duration.

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.