expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

RABBIT

TOKEN OVERVIEW

Risk Findings

Severity	Found
High	1
Medium	0
Low	5
Informational	0

Centralization Risks

Owner Privileges	Description
Can Owner Set Taxes >25% ?	Not Detected
Owner needs to enable trading ?	Yes, owner needs to enable trades
Can Owner Disable Trades ?	Not Detected
Can Owner Mint ?	Not Detected
Can Owner Blacklist ?	Not Detected
Can Owner set Max Wallet amount ?	Not Detected
Can Owner Set Max TX amount ?	Not Detected

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 with High Risk
KYC Verification	-
Audit Date	**8 June 2023**

CONTRACT DETAILS

Token Name: RABBIT

Symbol: RBT

Network: Ethereum

Language: Solidity

Contract Address: 0xdad36A6dbfA4DB527B80dB8b28258789E7eF3310

Total Supply: 8888888888888

Owner's Wallet: 0x8f4C3EAEA3B74e3d2F085380E97af9C34Fcb64FB

Deployer's Wallet:0x8f4C3EAEA3B74e3d2F085380E97af9C34Fcb64FB

Testnet Link:
https://testnet.bscscan.com/address/0x26a47c3f9f657fe0a7a5ccbcd465bdc8022d8c9c

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.

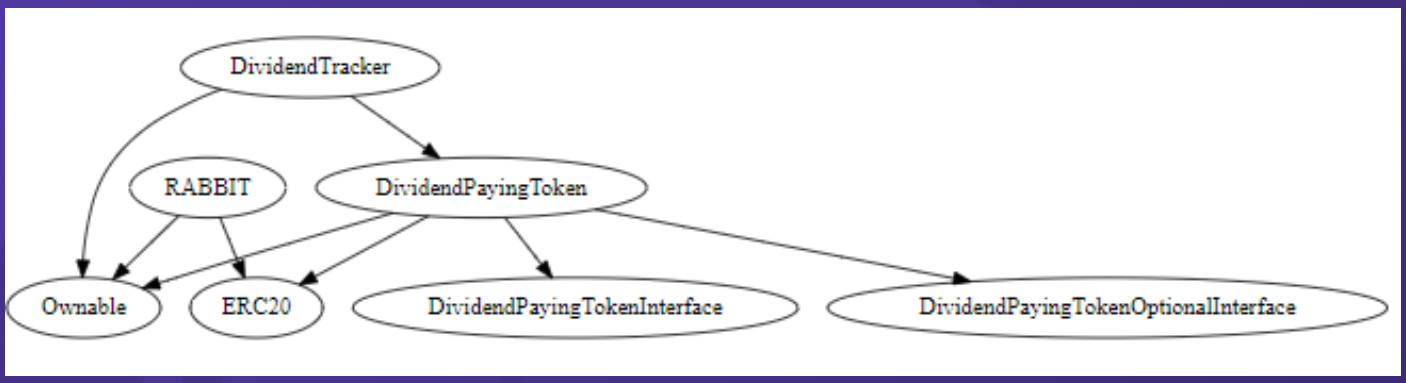
Low Risk

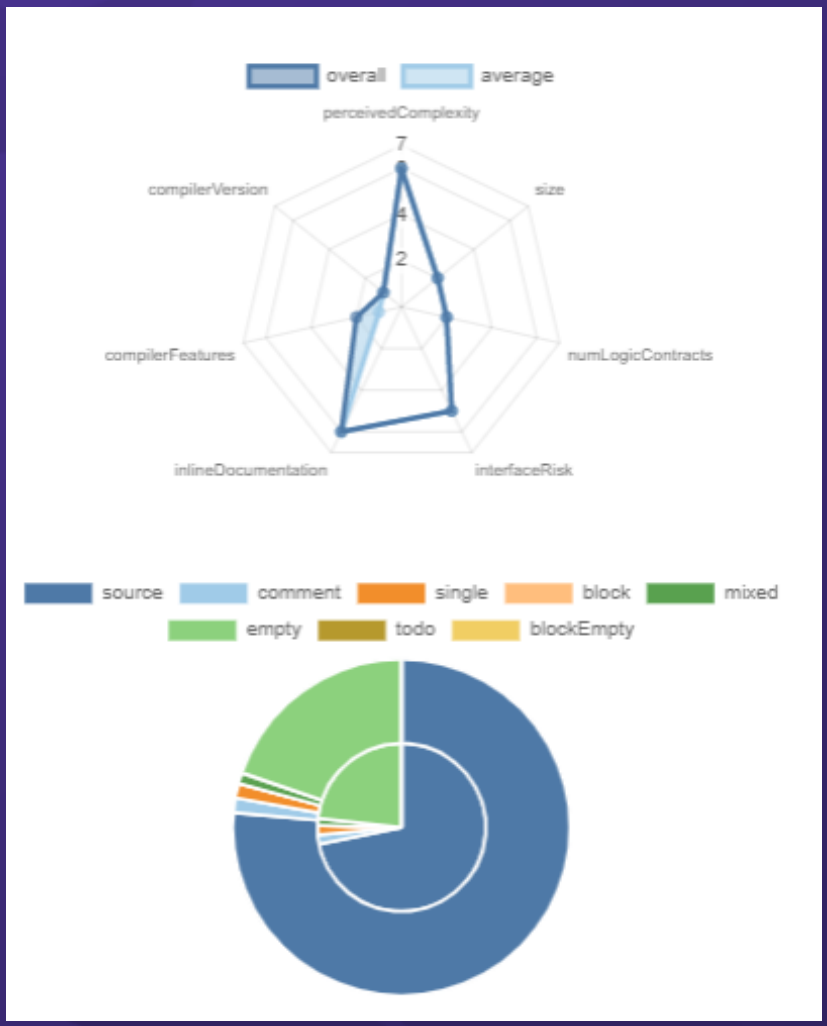
Issues on this level are minor details and warning that can remain unfixed.

Informational

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

INHERITANCE TREES





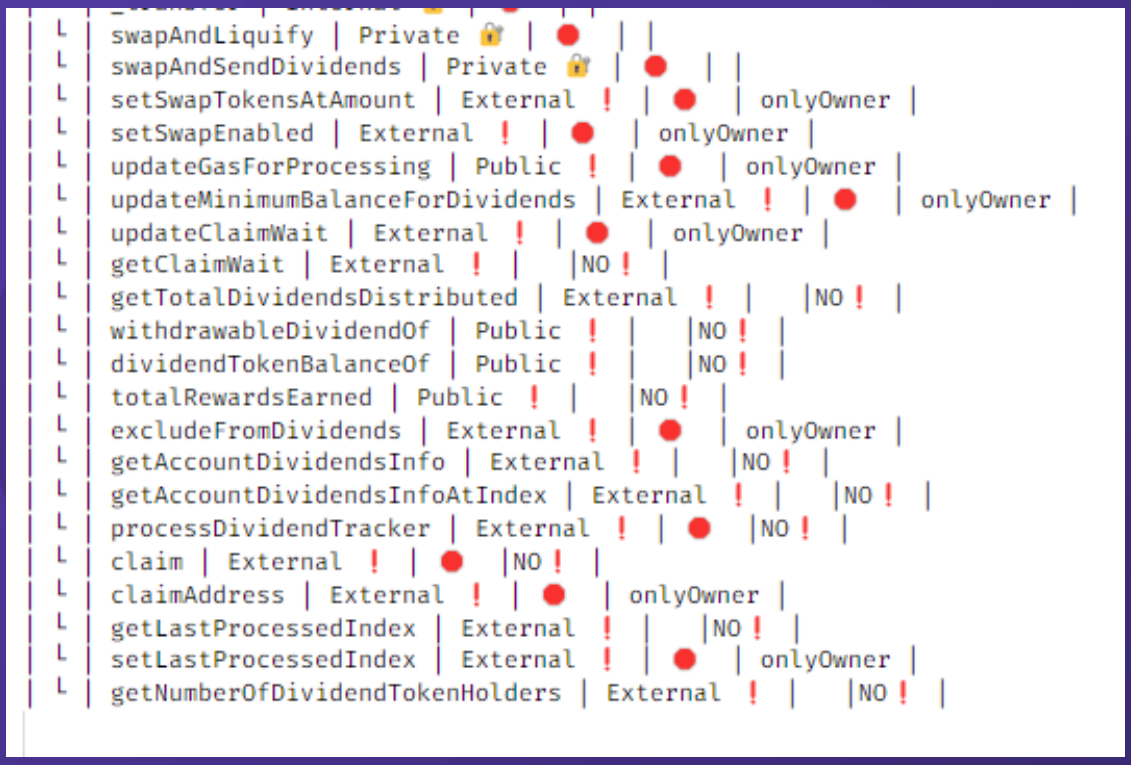
expelee

FUNCTION DETAILS

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 !	
└	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	
RABBIT	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	
└	changeMarketingWallet	External !	🔴	onlyOwner	
└	enableTrading	External !	🔴	onlyOwner	
└	_transfer	Internal 🔒	🔴		

expelee

FUNCTION DETAILS



└	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			

HIGH RISK FINDING

Trade must be enabled by the owner

Severty : High

Overview

Owner must enable trading to enable public to trade their tokens, otherwise no one would be able to buy /sell their tokens except whitelisted wallets.

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

Recommendation

To mitigate this issue you should enable trading before presale or transfer ownership to safu dev for initial days after presale.

LOW RISK FINDING

Owner can exclude account from fees

Severity : Low

Overview

Excludes/Includes an address from the collection of fees

```
function excludeFromFees(address account, bool excluded) external onlyOwner { //@audit-ok Owner
    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 exclude/include accounts from rewards

Severity : Low

Overview

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

```
ftrace | funcSig
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.

LOW RISK FINDING

Owner can change fees 5% at max

Severity : Low

Overview

Functions that allows the owner of the contract to update the buy/sell fees of the contract. For buy fees and sell fees maximum limit of 5%.

```
function updateBuyFees(uint256 _liquidityFeeOnBuy↑, uint256 _marketingFeeOnBuy↑, uint256 _rewardsFeeOnBuy↑, uint256 _trueBurnFeeOnBuy↑) external onlyOwner {
    liquidityFeeOnBuy   = _liquidityFeeOnBuy↑;
    marketingFeeOnBuy    = _marketingFeeOnBuy↑;
    rewardsFeeOnBuy     = _rewardsFeeOnBuy↑;
    trueBurnFeeOnBuy    = _trueBurnFeeOnBuy↑;

    totalBuyFee         = liquidityFeeOnBuy + marketingFeeOnBuy + rewardsFeeOnBuy + trueBurnFeeOnBuy;

    require(totalBuyFee ⩽ 500, "Buy fee cannot be more than 5%");

    emit BuyFeesUpdated(totalBuyFee);
} //@audit-issue Owner can change fees 5% max

ftrace | funcSig
function updateSellFees(uint256 _liquidityFeeOnSell↑, uint256 _marketingFeeOnSell↑, uint256 _rewardsFeeOnSell↑, uint256 _trueBurnFeeOnSell↑) external onlyOwner {
    liquidityFeeOnSell   = _liquidityFeeOnSell↑;
    marketingFeeOnSell    = _marketingFeeOnSell↑;
    rewardsFeeOnSell     = _rewardsFeeOnSell↑;
    trueBurnFeeOnSell    = _trueBurnFeeOnSell↑;

    totalSellFee        = liquidityFeeOnSell + marketingFeeOnSell + rewardsFeeOnSell + trueBurnFeeOnSell;

    require(totalSellFee ⩽ 500, "Sell fee cannot be more than 5%");

    emit SellFeesUpdated(totalSellFee);
}
```

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.

LOW RISK FINDING

Owner can change swap setting

Severity : Low

Overview

setSwapTokensAtAmount function allows the owner of the contract to update the value of **swapTokensAtAmount**. **toggleSwapping** function allows the contract owner to **enable** or **disable** the automatic **swapping**.

```
function setSwapTokensAtAmount(uint256 newAmount↑) external onlyOwner{ //@audit-ok Owner can change swap settings
    require(newAmount↑ > totalSupply() / 100_000, "SwapTokensAtAmount must be greater than 0.001% of total supply");
    swapTokensAtAmount = newAmount↑;
}

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

Recommendation

If the threshold is set too low, it could result in frequent and unnecessary swaps, which would increase gas fees and potentially lead to losses due to slippage. On the other hand, if the threshold is set too high, it could result in liquidity being insufficient to handle large trades, which could negatively impact the token price and liquidity pool. Be ensure that the contract owner account is well secured and only accessible by authorized parties.

LOW RISK FINDING

Owner can claim stuck tokens except native token

Severity : Low

Overview

Allows the contract owner to withdraw locked ERC20 tokens from the contract. The functions are properly restricted to only be executed by the contract owner.

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

Recommendation

While the functions are currently restricted to only be called by the contract owner, it is recommended to consider implementing a more robust access control mechanism.

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.