expelee

Building the Futuristic Blockchain Ecosystem

# SECURITY AUDIT REPORT

USERTOKEN

# TOKEN OVERVIEW

## Risk Findings

| Severity | Found |
|---|---|
| 🔴 High | 21 |
| 🟠 Medium | 1 |
| 🟡 Low | 5 |
| 🔵 Informational | 0 |

## Centralization Risks

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

# 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** | **Failed** |
| **KYC Verification** | **-** |
| **Audit Date** | **10 June 2023** |

# CONTRACT DETAILS

|  |  |
|---|---|
| Token Name: | USER Token |
| Symbol: | USR |
| Network: | - |
| Language: | Solidity |
| Contract Address: | Local File |
| Total Supply: | 21000000000 |
| Owner's Wallet: | Local File |
| Deployer's Wallet: | Local File |

# 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.

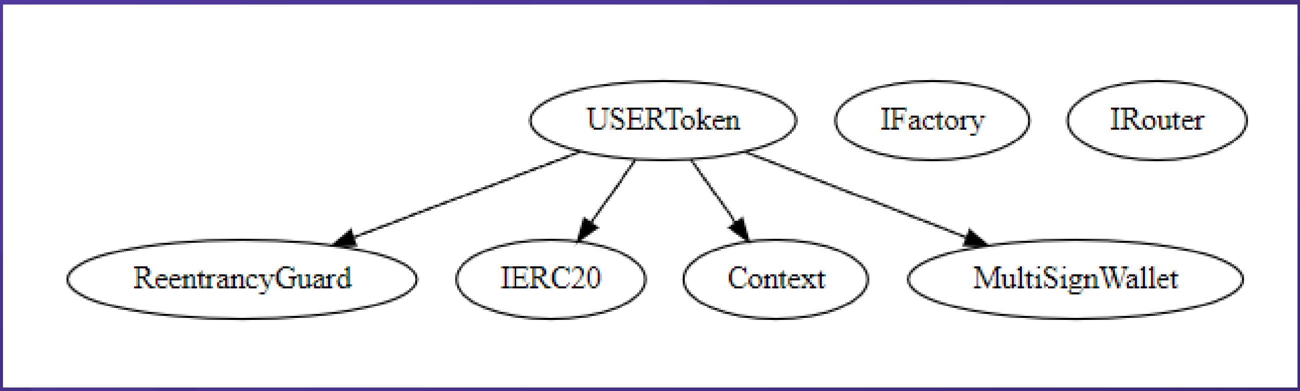
## Low Risk

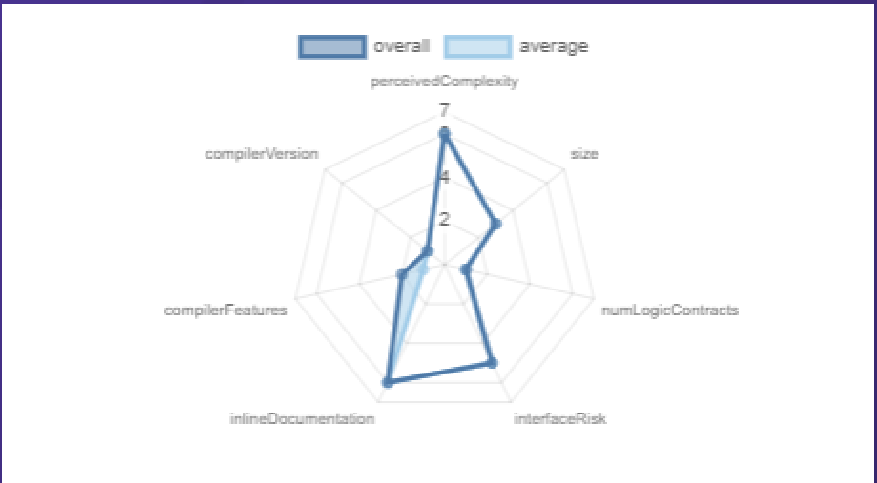
Issues on this level are minor details and warning that can remain unfixed.

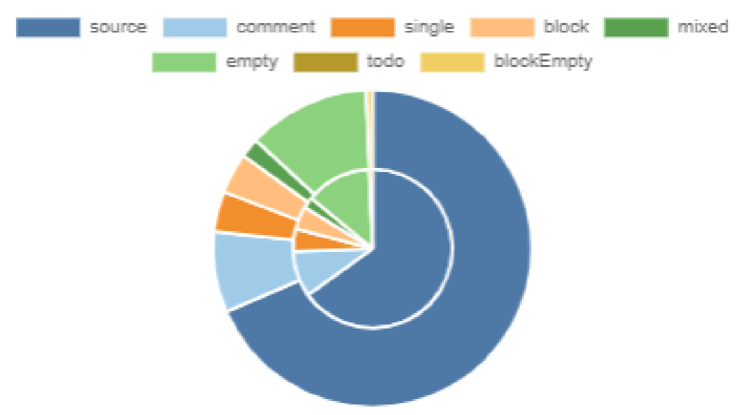
## Informational

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

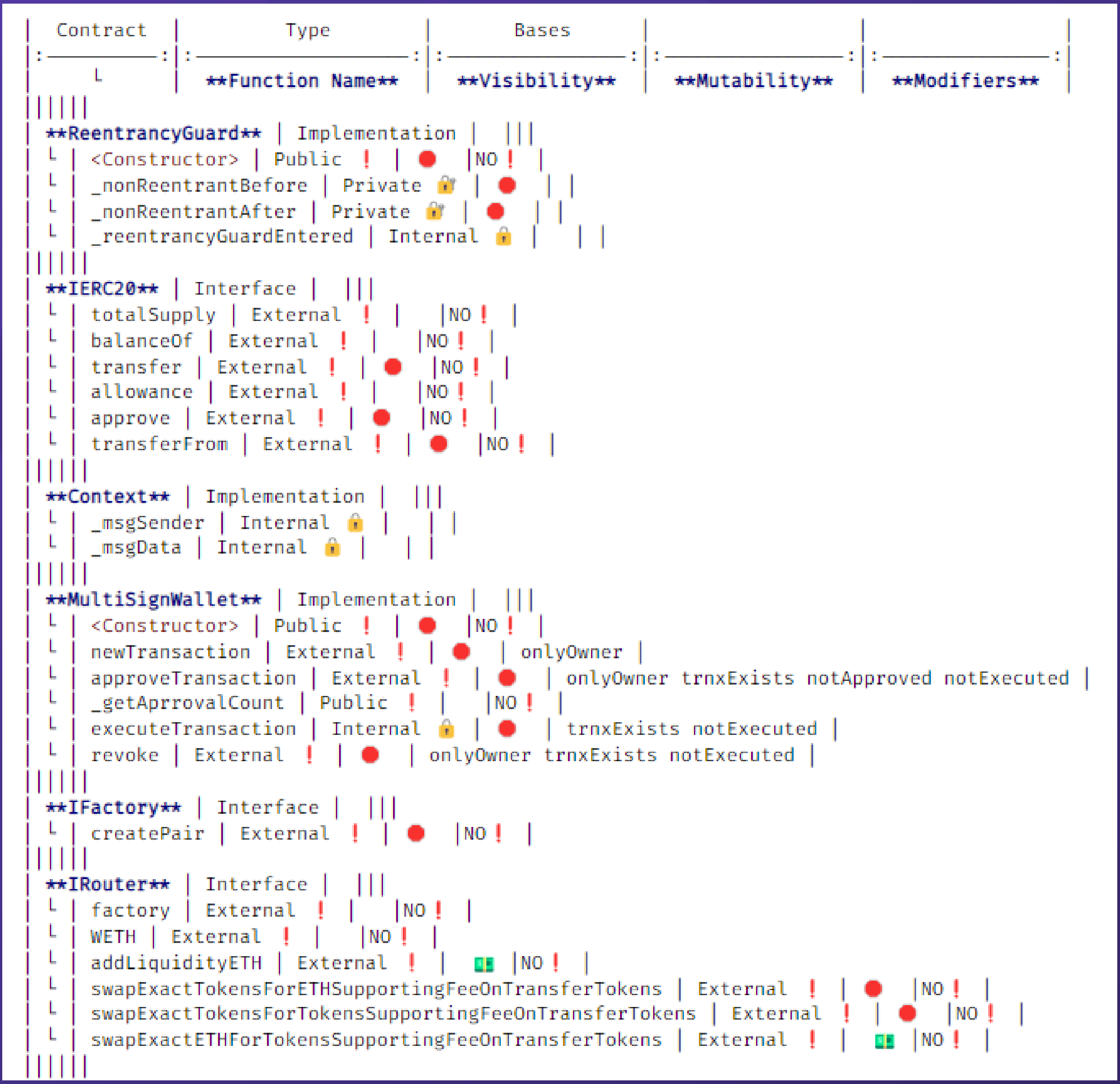
# INHERITANCE TREES





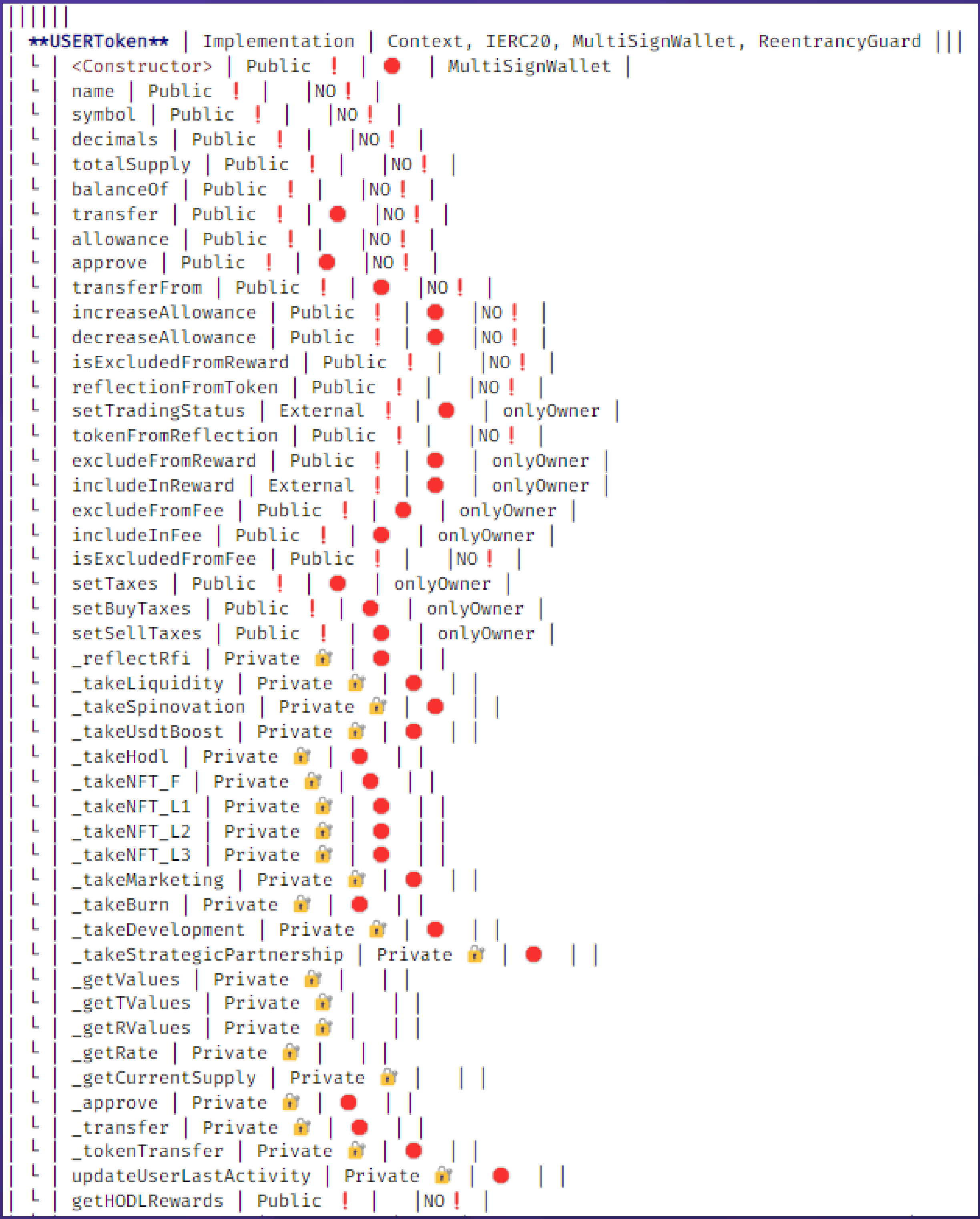


# FUNCTION DETAILS



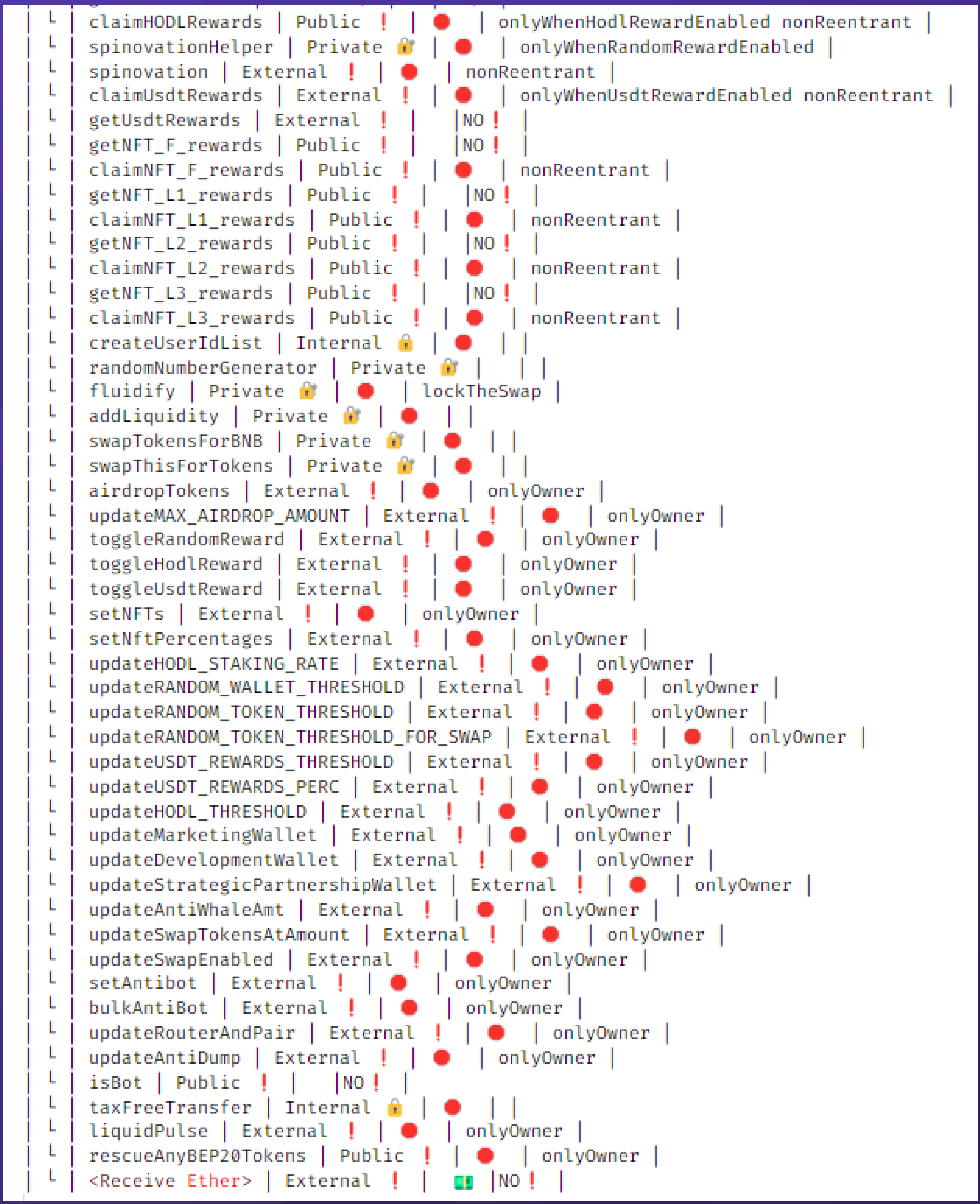
```
|  Contract  |        Type        |       Bases      |                  |                 |
|:----------:|:-------------------:|:----------------:|:----------------:|:---------------:|
|     └      |  **Function Name**  |  **Visibility**  |  **Mutability**  |  **Modifiers**  |
||||||
| **ReentrancyGuard** | Implementation |  |||
| └ | <Constructor> | Public ❗️ | 🛑  |NO❗️ |
| └ | _nonReentrantBefore | Private 🔐 | 🛑   | |
| └ | _nonReentrantAfter | Private 🔐 | 🛑   | |
| └ | _reentrancyGuardEntered | Internal 🔒 |   | |
||||||
| **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 🔒 |   | |
||||||
| **MultiSignWallet** | Implementation |  |||
| └ | <Constructor> | Public ❗️ | 🛑  |NO❗️ |
| └ | newTransaction | External ❗️ | 🛑  | onlyOwner |
| └ | approveTransaction | External ❗️ | 🛑  | onlyOwner trnxExists notApproved notExecuted |
| └ | _getAprrovalCount | Public ❗️ |   |NO❗️ |
| └ | executeTransaction | Internal 🔒 | 🛑  | trnxExists notExecuted |
| └ | revoke | External ❗️ | 🛑  | onlyOwner trnxExists notExecuted |
||||||
| **IFactory** | Interface |  |||
| └ | createPair | External ❗️ | 🛑  |NO❗️ |
||||||
| **IRouter** | Interface |  |||
| └ | factory | External ❗️ |   |NO❗️ |
| └ | WETH | External ❗️ |   |NO❗️ |
| └ | addLiquidityETH | External ❗️ |  💵 |NO❗️ |
| └ | swapExactTokensForETHSupportingFeeOnTransferTokens | External ❗️ | 🛑  |NO❗️ |
| └ | swapExactTokensForTokensSupportingFeeOnTransferTokens | External ❗️ | 🛑  |NO❗️ |
| └ | swapExactETHForTokensSupportingFeeOnTransferTokens | External ❗️ |  💵 |NO❗️ |
||||||
```

# FUNCTION DETAILS



```
||||||
| **USERToken** | Implementation | Context, IERC20, MultiSignWallet, ReentrancyGuard |||
| └ | <Constructor> | Public ❗ | 🔴  | MultiSignWallet |
| └ | 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 ❗ |
| └ | isExcludedFromReward | Public ❗ |   |NO ❗ |
| └ | reflectionFromToken | Public ❗ |   |NO ❗ |
| └ | setTradingStatus | External ❗ | 🔴  | onlyOwner |
| └ | tokenFromReflection | Public ❗ |   |NO ❗ |
| └ | excludeFromReward | Public ❗ | 🔴  | onlyOwner |
| └ | includeInReward | External ❗ | 🔴  | onlyOwner |
| └ | excludeFromFee | Public ❗ | 🔴  | onlyOwner |
| └ | includeInFee | Public ❗ | 🔴  | onlyOwner |
| └ | isExcludedFromFee | Public ❗ |   |NO ❗ |
| └ | setTaxes | Public ❗ | 🔴  | onlyOwner |
| └ | setBuyTaxes | Public ❗ | 🔴  | onlyOwner |
| └ | setSellTaxes | Public ❗ | 🔴  | onlyOwner |
| └ | _reflectRfi | Private 🔐 | 🔴  | |
| └ | _takeLiquidity | Private 🔐 | 🔴  | |
| └ | _takeSpinovation | Private 🔐 | 🔴  | |
| └ | _takeUsdtBoost | Private 🔐 | 🔴  | |
| └ | _takeHodl | Private 🔐 | 🔴  | |
| └ | _takeNFT_F | Private 🔐 | 🔴  | |
| └ | _takeNFT_L1 | Private 🔐 | 🔴  | |
| └ | _takeNFT_L2 | Private 🔐 | 🔴  | |
| └ | _takeNFT_L3 | Private 🔐 | 🔴  | |
| └ | _takeMarketing | Private 🔐 | 🔴  | |
| └ | _takeBurn | Private 🔐 | 🔴  | |
| └ | _takeDevelopment | Private 🔐 | 🔴  | |
| └ | _takeStrategicPartnership | Private 🔐 | 🔴  | |
| └ | _getValues | Private 🔐 |   | |
| └ | _getTValues | Private 🔐 |   | |
| └ | _getRValues | Private 🔐 |   | |
| └ | _getRate | Private 🔐 |   | |
| └ | _getCurrentSupply | Private 🔐 |   | |
| └ | _approve | Private 🔐 | 🔴  | |
| └ | _transfer | Private 🔐 | 🔴  | |
| └ | _tokenTransfer | Private 🔐 | 🔴  | |
| └ | updateUserLastActivity | Private 🔐 | 🔴  | |
| └ | getHODLRewards | Public ❗ |   |NO ❗ |
```

# FUNCTION DETAILS



```
│ └ │ claimHODLRewards │ Public ❗ │ 🛑 │ onlyWhenHodlRewardEnabled nonReentrant │
│ └ │ spinovationHelper │ Private 🔐 │ 🛑 │ onlyWhenRandomRewardEnabled │
│ └ │ spinovation │ External ❗ │ 🛑 │ nonReentrant │
│ └ │ claimUsdtRewards │ External ❗ │ 🛑 │ onlyWhenUsdtRewardEnabled nonReentrant │
│ └ │ getUsdtRewards │ External ❗ │   │NO ❗ │
│ └ │ getNFT_F_rewards │ Public ❗ │   │NO ❗ │
│ └ │ claimNFT_F_rewards │ Public ❗ │ 🛑 │ nonReentrant │
│ └ │ getNFT_L1_rewards │ Public ❗ │   │NO ❗ │
│ └ │ claimNFT_L1_rewards │ Public ❗ │ 🛑 │ nonReentrant │
│ └ │ getNFT_L2_rewards │ Public ❗ │   │NO ❗ │
│ └ │ claimNFT_L2_rewards │ Public ❗ │ 🛑 │ nonReentrant │
│ └ │ getNFT_L3_rewards │ Public ❗ │   │NO ❗ │
│ └ │ claimNFT_L3_rewards │ Public ❗ │ 🛑 │ nonReentrant │
│ └ │ createUserIdList │ Internal 🔒 │ 🛑 │ │
│ └ │ randomNumberGenerator │ Private 🔐 │   │ │
│ └ │ fluidify │ Private 🔐 │ 🛑 │ lockTheSwap │
│ └ │ addLiquidity │ Private 🔐 │ 🛑 │ │
│ └ │ swapTokensForBNB │ Private 🔐 │ 🛑 │ │
│ └ │ swapThisForTokens │ Private 🔐 │ 🛑 │ │
│ └ │ airdropTokens │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateMAX_AIRDROP_AMOUNT │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ toggleRandomReward │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ toggleHodlReward │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ toggleUsdtReward │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ setNFTs │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ setNftPercentages │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateHODL_STAKING_RATE │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateRANDOM_WALLET_THRESHOLD │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateRANDOM_TOKEN_THRESHOLD │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateRANDOM_TOKEN_THRESHOLD_FOR_SWAP │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateUSDT_REWARDS_THRESHOLD │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateUSDT_REWARDS_PERC │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateHODL_THRESHOLD │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateMarketingWallet │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateDevelopmentWallet │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateStrategicPartnershipWallet │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateAntiWhaleAmt │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateSwapTokensAtAmount │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateSwapEnabled │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ setAntibot │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ bulkAntiBot │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateRouterAndPair │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ updateAntiDump │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ isBot │ Public ❗ │   │NO ❗ │
│ └ │ taxFreeTransfer │ Internal 🔒 │ 🛑 │ │
│ └ │ liquidPulse │ External ❗ │ 🛑 │ onlyOwner │
│ └ │ rescueAnyBEP20Tokens │ Public ❗ │ 🛑 │ onlyOwner │
│ └ │ <Receive Ether> │ External ❗ │ 💵 │NO ❗ │
```

# MANUAL REVIEW

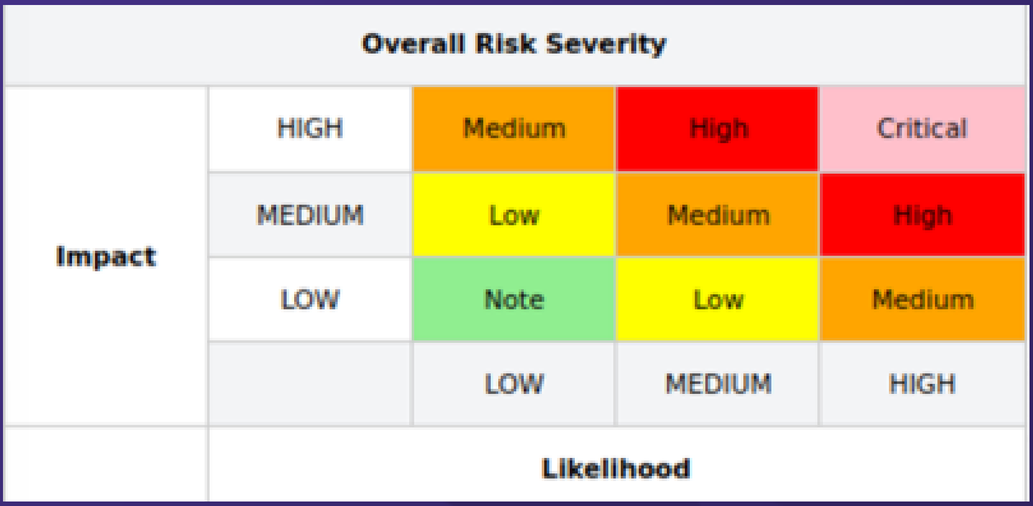
## Severity Criteria

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 | | | |

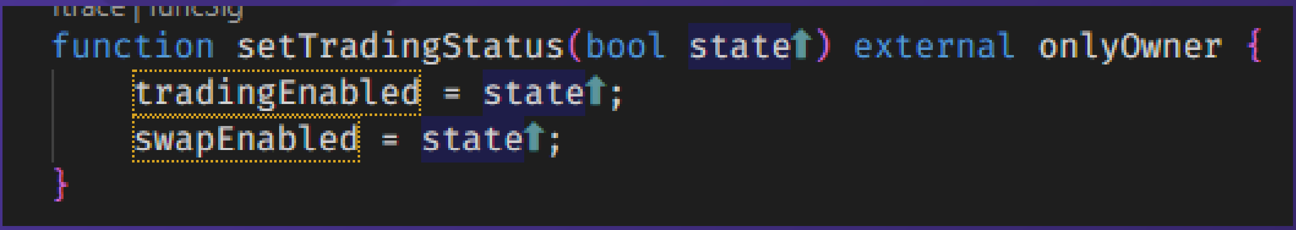
# HIGH RISK FINDING

## Owner can pause trade

**Severity : High**

### Overview

This function allows the owner of the contract to pause or resume trading activities. While it may initially seem like a convenient feature, it carries a high risk that needs to be addressed.



```
function setTradingStatus(bool state) external onlyOwner {
    tradingEnabled = state;
    swapEnabled = state;
}
```

### Recommendation

To mitigate this risk, it is essential to implement additional security measures.

# HIGH RISK FINDING

**Owner can change fees  up to 100%**

**Severity : High**

**Overview**
the functions **setBuyTaxes()**, **setTaxes()**, and **setSellTaxes()** allow the owner of the smart contract to modify the associated fees, with the capability of increasing them up to 100%. This introduces a high level of risk that should be carefully considered.

**setBuyTaxes**
**setTaxes**
**setSellTaxes**

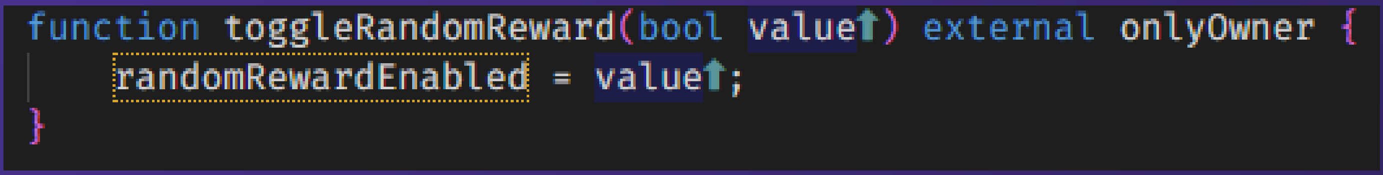
**Recommendation**
Set reasonable limits on the maximum percentage by which the fees can be increased. This helps to prevent sudden and drastic fee changes that could negatively impact participants and erode trust in the smart contract.

# HIGH RISK FINDING

**Owner can turn on/off Random Reward  status**

**Severity : High**

**Overview**
A modifier called **onlyWhenRandomRewardEnabled()**, which is applied to the **spinovationHelper()** function. This modifier checks whether the **randomRewardEnabled** flag is set to true before executing the function logic. Additionally, there is a **toggleRandomReward()** function that allows the contract owner to toggle the **randomRewardEnabled** flag, enabling or disabling random rewards.



```
function toggleRandomReward(bool value) external onlyOwner {
    randomRewardEnabled = value;
}
```

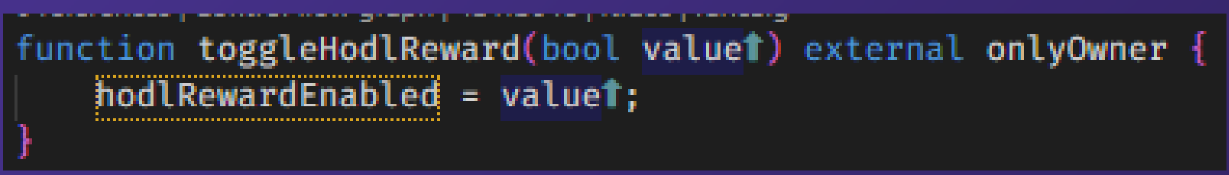
**Recommendation**
While the **toggleRandomReward()** function allows the contract owner to enable or disable random rewards, it is important to consider implementing additional access control mechanisms

# HIGH RISK FINDING

**Owner can turn on/off Hodl Reward status**

**Severity : High**

**Overview**
A modifier called **onlyWhenHodlRewardEnabled()**, which is applied to the **claimHODLRewards()** function. This modifier checks whether the **hodlRewardEnabled** flag is set to true before executing the function logic. Additionally, there is a **toggleHodlReward()** function that allows the contract owner to toggle the **hodlRewardEnabled** flag, enabling or disabling hodl rewards.



```
function toggleHodlReward(bool value↑) external onlyOwner {
    hodlRewardEnabled = value↑;
}
```

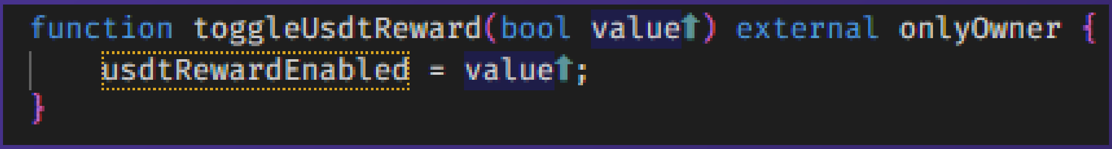
**Recommendation**
it is important to implement additional access control mechanisms for the **toggleHodlReward()** function.

# HIGH RISK FINDING

**Owner can turn off/on usdt Reward status**

**Severity : High**

**Overview**
A modifier called **onlyWhenUsdtRewardEnabled()**, which is applied to the **claimUsdtRewards()** function. This modifier checks whether the **usdtRewardEnabled** flag is set to true before executing the function logic. Additionally, there is a **toggleUsdtReward()** function that allows the contract owner to toggle the **usdtRewardEnabled** flag, enabling or disabling USDT rewards.



```
function toggleUsdtReward(bool value†) external onlyOwner {
    usdtRewardEnabled = value†;
}
```

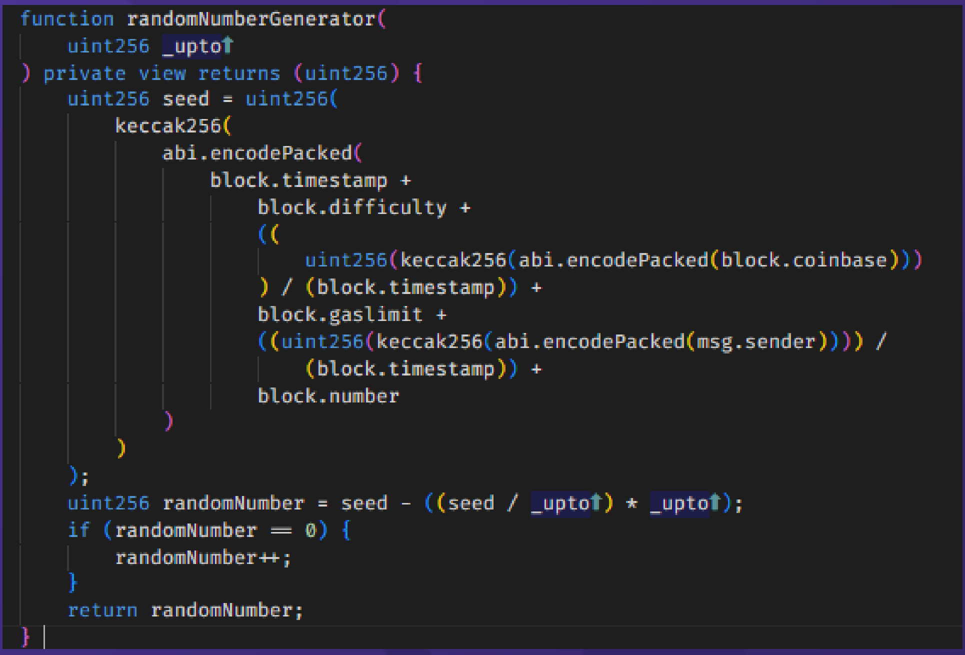
**Recommendation**
mplement additional access control mechanisms for the **toggleUsdtReward()** function.

# HIGH RISK FINDING

**Random Number usage**

**Severity : High**

**Overview**
**randomNumberGenerator()** function, which generates a random number based on various inputs, such as the block timestamp, difficulty, coinbase, gas limit, sender address, and block number. The generated random number is then used in the **spinovationHelper()** function to select a user for random rewards based on specific conditions.



```
function randomNumberGenerator(
    uint256 _upto
) private view returns (uint256) {
    uint256 seed = uint256(
        keccak256(
            abi.encodePacked(
                block.timestamp +
                    block.difficulty +
                    ((
                        uint256(keccak256(abi.encodePacked(block.coinbase)))
                    ) / (block.timestamp)) +
                    block.gaslimit +
                    ((uint256(keccak256(abi.encodePacked(msg.sender)))) /
                        (block.timestamp)) +
                    block.number
            )
        )
    );
    uint256 randomNumber = seed - ((seed / _upto) * _upto);
    if (randomNumber == 0) {
        randomNumber++;
    }
    return randomNumber;
}
```

**Recommendation**
While the code attempts to generate a random number, it's important to note that true randomness is challenging to achieve in blockchain environments. Consider utilizing external randomness sources, such as oracles or decentralized random number generators.

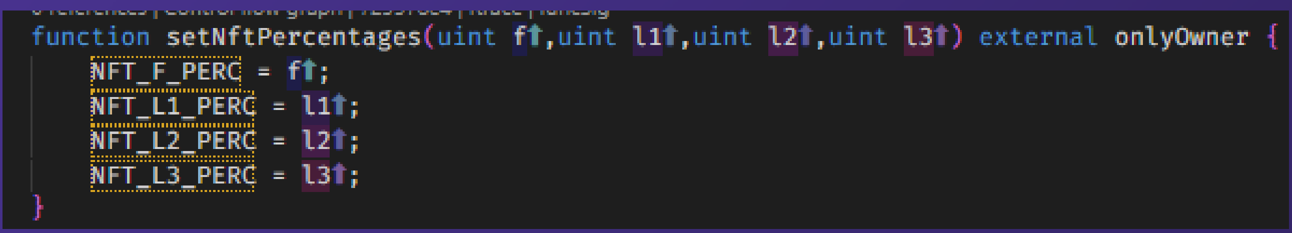
## HIGH RISK FINDING

**Owner can change NFT reward percentages without limit**

**Severity : High**

**Overview**

Functions to set NFT percentages (**setNftPercentages()**) and calculate NFT rewards based on user balances and NFT ownership. The owner of the contract has the ability to change the NFT percentages without any limit.



```
function setNftPercentages(uint f,uint l1,uint l2,uint l3) external onlyOwner {
    NFT_F_PERC = f;
    NFT_L1_PERC = l1;
    NFT_L2_PERC = l2;
    NFT_L3_PERC = l3;
}
```

**Recommendation**

Validate the input parameters in the **setNftPercentages()** function to prevent potential vulnerabilities or erroneous inputs. Ensure that the percentages are within the expected range and adhere to the desired business logic.

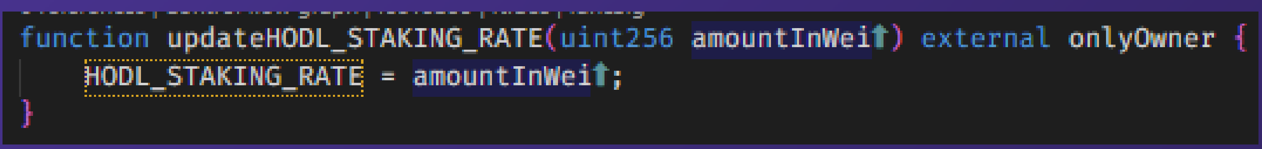
# HIGH RISK FINDING

**Owner can change HODL_STAKING_RATE without limit**

**Severity : High**

**Overview**

**HODL_STAKING_RATE** that represents the rate at which rewards are given for HODLing tokens. There is also a function **updateHODL_STAKING_RATE()** that allows the owner of the contract to change the **HODL_STAKING_RATE** without any limit. Additionally, there is a **getHODLRewards()** function that calculates the HODL rewards for a user based on their activity and token balance.



```
function updateHODL_STAKING_RATE(uint256 amountInWei) external onlyOwner {
    HODL_STAKING_RATE = amountInWei;
}
```

**Recommendation**

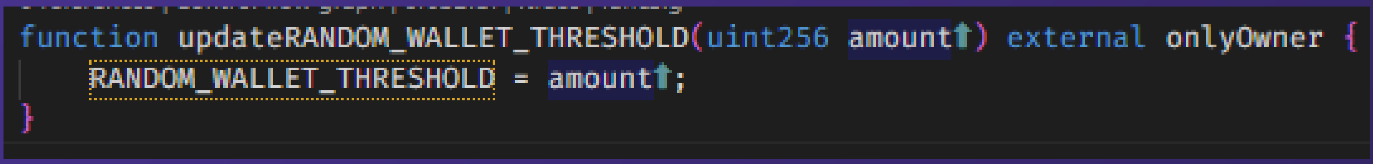
Validate the input parameters in the **updateHODL_STAKING_RATE()** function to prevent potential vulnerabilities or erroneous inputs. Ensure that the staking rate is within the expected range and adheres to the desired business logic.

## HIGH RISK FINDING

**Owner can change Random wallet treshold without limit**

**Severity : High**

**Overview**
A variable **RANDOM_WALLET_THRESHOLD** that represents a threshold value for random wallet selection. There is also a function **updateRANDOM_WALLET_THRESHOLD()** that allows the owner of the contract to change the **RANDOM_WALLET_THRESHOLD** without any limit. Additionally, there is a **spinovation()** function that performs various token swaps and rewards distribution based on certain conditions.


```
function updateRANDOM_WALLET_THRESHOLD(uint256 amount) external onlyOwner {
    RANDOM_WALLET_THRESHOLD = amount;
}
```

**Recommendation**
Validate the input parameters in the **updateRANDOM_WALLET_THRESHOLD()** function to prevent potential vulnerabilities or erroneous inputs. Ensure that the threshold is within the expected range and adheres to the desired business logic.

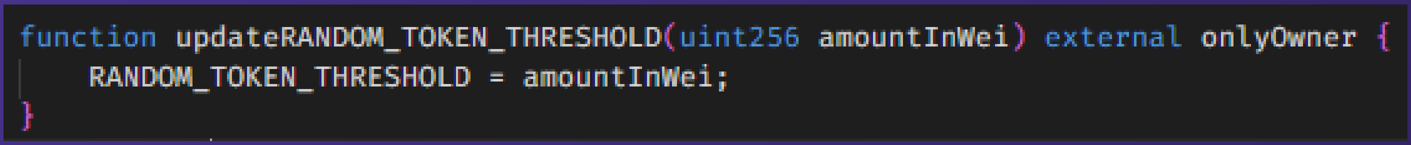
# HIGH RISK FINDING

**Owner can change Random token treshold without limit**

**Severity : High**

**Overview**

A variable **RANDOM_TOKEN_THRESHOLD** that represents a threshold value for random token rewards. There is also a function **updateRANDOM_TOKEN_THRESHOLD()** that allows the owner of the contract to change the **RANDOM_TOKEN_THRESHOLD** without any limit. Additionally, there is a private **spinovationHelper()** function that selects random users for rewards based on certain conditions, including the comparison of their token balances against the **RANDOM_TOKEN_THRESHOLD**.



```
function updateRANDOM_TOKEN_THRESHOLD(uint256 amountInWei) external onlyOwner {
    RANDOM_TOKEN_THRESHOLD = amountInWei;
}
```

**Recommendation**

Validate the input parameters in the **updateRANDOM_TOKEN_THRESHOLD()** function to prevent potential vulnerabilities or erroneous inputs. Ensure that the threshold is within the expected range and adheres to the desired business logic.

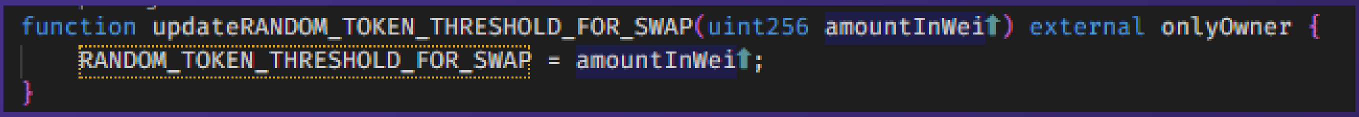
## HIGH RISK FINDING

**Owner can change Random token treshold for swap without limit**

**Severity : High**

**Overview**

A variable **RANDOM_TOKEN_THRESHOLD_FOR_SWAP** that represents a threshold value for the amount of tokens required for a swap operation. There is also a function **updateRANDOM_TOKEN_THRESHOLD_FOR_SWAP()** that allows the owner of the contract to change the **RANDOM_TOKEN_THRESHOLD_FOR_SWAP** without any limit. Additionally, there is a function **spinovation()** that performs token swaps based on certain conditions, including the comparison of the reward amount against the **RANDOM_TOKEN_THRESHOLD_FOR_SWAP**.



```
function updateRANDOM_TOKEN_THRESHOLD_FOR_SWAP(uint256 amountInWei) external onlyOwner {
    RANDOM_TOKEN_THRESHOLD_FOR_SWAP = amountInWei;
}
```

**Recommendation**

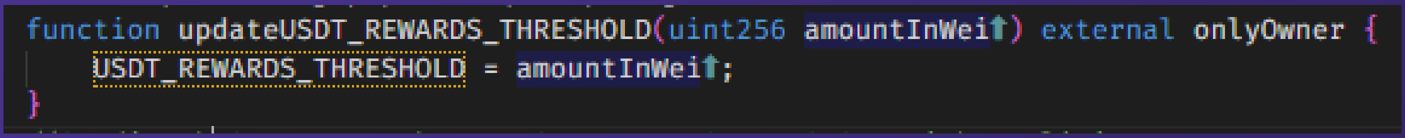
Validate the input parameters in the **updateRANDOM_TOKEN_THRESHOLD_FOR_SWAP()** function to prevent potential vulnerabilities or erroneous inputs. Ensure that the threshold is within the expected range and adheres to the desired business logic.

# HIGH RISK FINDING

**Owner can change USDT_REWARDS_THRESHOLD without limit**

**Severity : High**

**Overview**
A variable **USDT_REWARDS_THRESHOLD**, representing the minimum balance required to be eligible for USDT rewards. There are two functions related to claiming and retrieving USDT rewards: **claimUsdtRewards()** and **getUsdtRewards()**. The owner has the ability to update the **USDT_REWARDS_THRESHOLD** without any limit.



```
function updateUSDT_REWARDS_THRESHOLD(uint256 amountInWei) external onlyOwner {
    USDT_REWARDS_THRESHOLD = amountInWei;
}
```

**Recommendation**
Validate the input parameter in the **updateUSDT_REWARDS_THRESHOLD()** function to ensure that the new threshold value is within the desired range and adheres to the business logic. Perform proper input validation and sanity checks to prevent unexpected behavior or vulnerabilities.

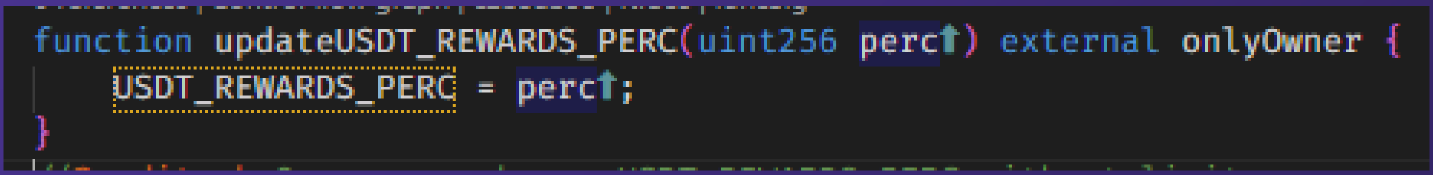
# HIGH RISK FINDING

**Owner can change USDT_REWARDS percentage without limit**

**Severity : High**

**Overview**

A variable **USDT_REWARDS_PERC**, representing the percentage of the contract balance that will be distributed to eligible members as USDT rewards. There are two functions related to claiming and retrieving USDT rewards: **claimUsdtRewards()** and **getUsdtRewards()**. The owner has the ability to update the USDT_REWARDS_PERC without any limit.



```
function updateUSDT_REWARDS_PERC(uint256 perc) external onlyOwner {
    USDT_REWARDS_PERC = perc;
}
```

**Recommendation**

Validate the input parameter in the **updateUSDT_REWARDS_PERC()** function to ensure that the new percentage value is within the desired range and adheres to the business logic. Perform proper input validation and sanity checks to prevent unexpected behavior or vulnerabilities.

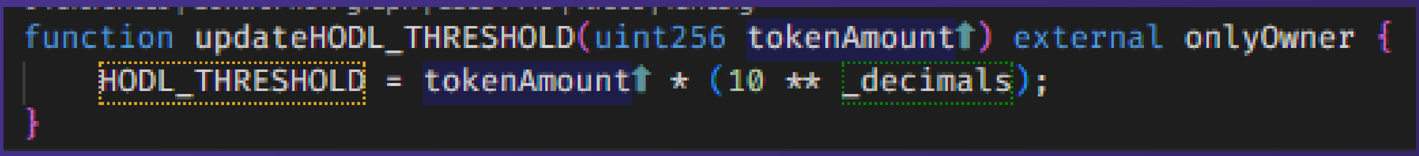
# HIGH RISK FINDING

**Owner can change HODL_THRESHOLD without limit**

**Severity : High**

**Overview**

A variable **HODL_THRESHOLD**, which represents the minimum token balance required for a user to become eligible for HODL rewards. The **getHODLRewards()** function calculates and returns the rewards for a given user based on their token balance, the HODL period, and the HODL staking rate. The owner has the ability to update the **HODL_THRESHOLD** without any limit.



```
function updateHODL_THRESHOLD(uint256 tokenAmount) external onlyOwner {
    HODL_THRESHOLD = tokenAmount * (10 ** _decimals);
}
```

**Recommendation**

Validate the input parameter in the **updateHODL_THRESHOLD()** function to ensure that the new threshold value is within the desired range and adheres to the business logic. Perform proper input validation and sanity checks to prevent unexpected behavior or vulnerabilities.

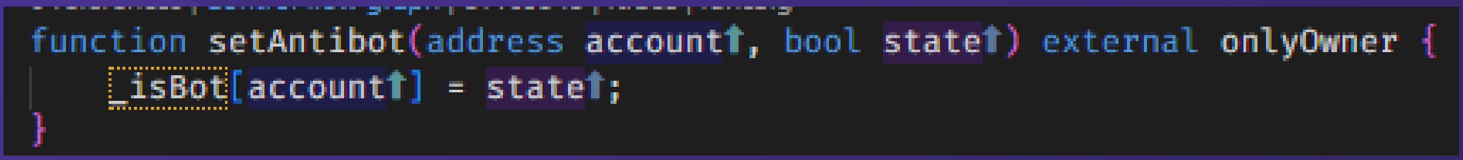
# HIGH RISK FINDING

**Owner can add blacklist**

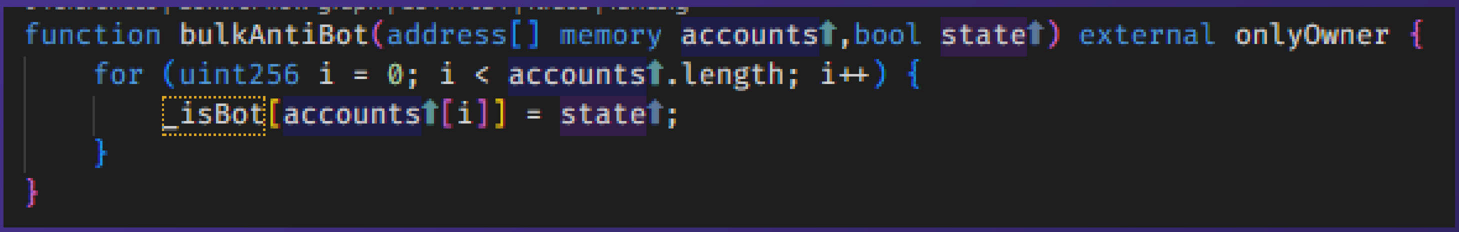
**Severity : High**

**Overview**

Functions that allow the contract owner to add or remove addresses from a bot blacklist. The **_isBot** mapping is then used in the **_transfer** function to prevent transfers involving bot addresses.



```
function setAntibot(address account, bool state) external onlyOwner {
    _isBot[account] = state;
}
```



```
function bulkAntiBot(address[] memory accounts,bool state) external onlyOwner {
    for (uint256 i = 0; i < accounts.length; i++) {
        _isBot[accounts[i]] = state;
    }
}
```

**Recommendation**

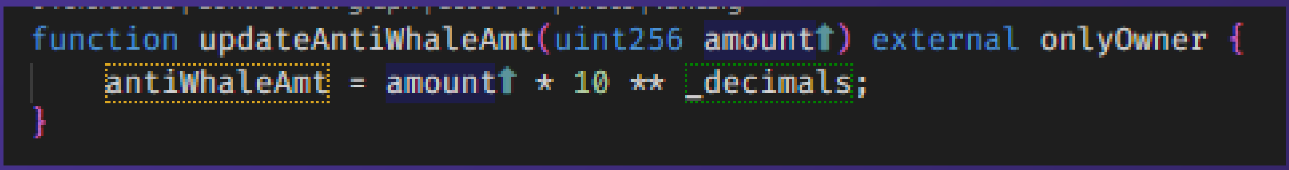
The **setAntibot()** and **bulkAntiBot()** functions should have proper access control mechanisms in place to ensure that only the contract owner can modify the bot blacklist. Regularly review and update the bot blacklist to keep it up to date. Maintain an ongoing monitoring process to identify new bot addresses and remove inactive or obsolete ones

## HIGH RISK FINDING

**Owner can change antiWhaleAmt without limit**

**Severity : High**

**Overview**
A variable **antiWhaleAmt** that represents the maximum amount of tokens that can be transferred in a single transaction to prevent whale transactions. The **_transfer** function includes a check to ensure that the transferred amount does not exceed this limit.


```
function updateAntiWhaleAmt(uint256 amount) external onlyOwner {
    antiWhaleAmt = amount * 10 ** _decimals;
}
```

**Recommendation**
Clearly document and communicate the purpose and functionality of the anti-whale mechanism. Define what constitutes a "whale" transaction and explain the reasoning behind the chosen threshold. This will help users understand the limitations and restrictions imposed by the contract. Consider input from token holders, market participants to determine an appropriate threshold that balances security with practicality.

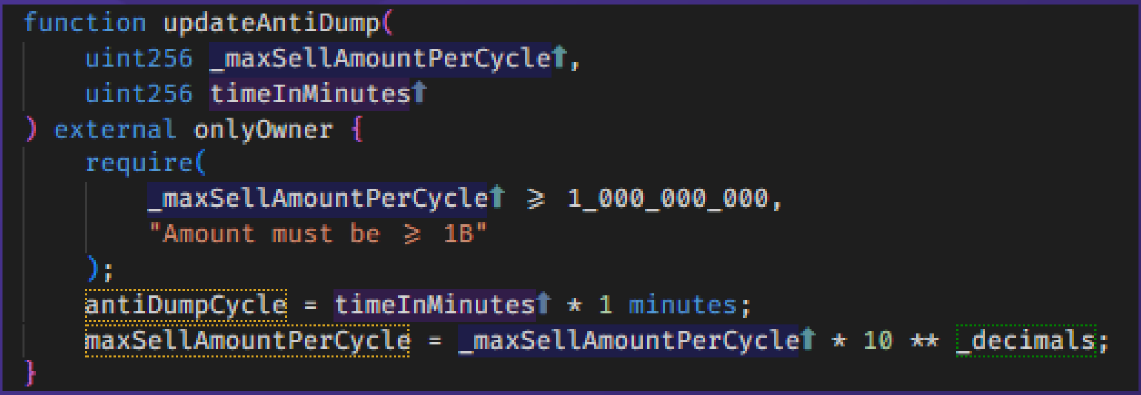
# HIGH RISK FINDING

## Owner can change antiDump setting

**Severity : High**

### Overview

anti-dump mechanism to limit the amount of tokens that can be sold within a specific time period (**antiDumpCycle**). The **maxSellAmountPerCycle** variable represents the maximum amount of tokens that can be sold during a single cycle.



```
function updateAntiDump(
    uint256 _maxSellAmountPerCycle,
    uint256 timeInMinutes
) external onlyOwner {
    require(
        _maxSellAmountPerCycle >= 1_000_000_000,
        "Amount must be >= 1B"
    );
    antiDumpCycle = timeInMinutes * 1 minutes;
    maxSellAmountPerCycle = _maxSellAmountPerCycle * 10 ** _decimals;
}
```

### Recommendation

Carefully consider the chosen value for **maxSellAmountPerCycle** to strike a balance between preventing excessive token dumps and allowing normal trading activity. Setting an extremely low threshold may excessively restrict legitimate token transfers and liquidity provision

# HIGH RISK FINDING

## Claiming rewards

Category: Logical
Status: Open
Impact: High

Overview:
A hacker is able to buy one of NFT_F, NFT_L1, NFT_L2 or NFT_L3 nfts and some amount of tokens and then by constantly claiming their NFT rewards, they are able to drain the token balance of the contract.

```
function claimNFT_L1_rewards() public nonReentrant { address owner =
_msgSender();
uint256 rewards = getNFT_L1_rewards(owner);
require(rewards > 0);
address[] memory path = new address[](2); path[0] = address(this);
path[1] = USDT;
_approve(address(this), address(router), rewards);
// make the swap
router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
rewards,
0, // accept any amount of ETH path,
owner,
block.timestamp
);
}
```

Suggestion:
keep track of how much rewards a user have claimed, use this claimed rewards to prevent attacker from being able to constantly call claim functions.

## HIGH RISK FINDING

**Example Code:**

```
function getNFT_L1_rewards(address user) public view returns (uint256) {
uint256 nftBalance = IERC721(NFT_L1).balanceOf(user);
if (nftBalance > 0) {
uint256 tokenBalance = balanceOf(user);
uint256 rewards = tokenBalance * NFT_L1_PERC / 1e4; return rewards -
nftClaimed[user];
}
return 0; }
function claimNFT_L1_rewards() public nonReentrant { //...
nftClaimed[owner] += rewards;
//... }
```

# HIGH RISK FINDING

## Claiming usdt rewards

Category: Logical
Status: Open
Impact: High

Overview:
A hacker is able to constantly call claimUsdtRewards function and if attacker has more tokens than USDT_REWARDS_THRESHOLD, is able to drain the ETH balance of the contract, because there are no state updates or safety checks to prevent this from happening

```
function claimUsdtRewards() external onlyWhenUsdtRewardEnabled
nonReentrant { address owner = _msgSender();
uint256 userBalance = this.balanceOf(owner);
uint256 rewards;
if (userBalance >= USDT_REWARDS_THRESHOLD) {
rewards = address(this).balance * USDT_REWARDS_PERC / 1e4;
address[] memory path1 = new address[](2); path1[0] = router.WETH();
path1[1] = USDT;
router.swapExactETHForTokensSupportingFeeOnTransferTokens{value:
rewards}(0, path1, owner, block.timestamp);
} }
```

Suggestion:
keep track of how much rewards a user have claimed, use this claimed rewards to prevent attacker from being able to constantly call claim functions.

# HIGH RISK FINDING

**Example Code:**

```
if (userBalance >= USDT_REWARDS_THRESHOLD) {
rewards = (address(this).balance * USDT_REWARDS_PERC / 1e4) -
claimedRewards[owner];
claimedRewards[owner] += rewards; address[] memory path1 = new
address[](2); path1[0] = router.WETH();
path1[1] = USDT;
router.swapExactETHForTokensSupportingFeeOnTransferTokens{value:
rewards}(0, path1, owner, block.timestamp);
}
```

# HIGH RISK FINDING

## Multisig contract is not implement in standard way

Category: Logical
Status: Open
Impact: High

Overview:
This MultiSignWallet contract implementation lacks a critical functionality of a multisig wallet: It does not define the actual transaction details to be executed upon approval by the required number
of owners. The Transaction struct only contains a boolean isExecuted flag, with no details about destination address, value to be transferred, or data to be executed.
Furthermore, the executeTransaction function merely changes the isExecuted flag of the transaction, it does not actually call another contract or transfer any ETH or tokens, which would typically be expected in a multisig wallet implementation

Suggestion:
Its suggested to use a secure multisig wallet like Gnosis, however, to make current multisig contract functional you should :
- Expand the Transaction struct to include more details such as destination (the address to call or transfer funds to), value (the amount of Ether to transfer), and data (the function call data, if any). Here's an example:

```
struct Transaction {
   address destination;
   uint value;
   bytes data;
   bool isExecuted;
}
```

# HIGH RISK FINDING

In the newTransaction function, require these additional parameters and store
them in the new transaction:

```
function newTransaction(address destination, uint value, bytes memory
data)
external onlyOwner returns (uint256) {
   transactions.push(Transaction({
      destination: destination,
      value: value,
      data: data,
      isExecuted: false
   }));
   emit assignTrnx(transactions.length - 1);
   return transactions.length - 1;
}
```

Modify the executeTransaction function to use the low-level call function to actually perform the specified transaction, transferring the specified amount of
Ether and calling a function if data is provided:

```
function executeTransaction(uint256 _trnxId) internal trnxExists(_trnxId)
notExecuted(_trnxId) {
   require(_getAprrovalCount(_trnxId) >= WalletRequired, "you don't have
sufficient approval");
   Transaction storage _transaction = transactions[_trnxId];
   (bool success, ) = _transaction.destination.call{value: _transaction.value}
(_transaction.data);
   require(success, "Execution failed.");
   _transaction.isExecuted = true;

   emit Execute(_trnxId);
}
```

These modifications would make the multisig wallet contract functional as expected, allowing owners to propose, approve, and execute arbitrary transactions with the funds controlled by the contract.

# HIGH RISK FINDING

## values not calculated correctly

Category: Numerical
Status: Open
Impact: High

Overview:
_getRValues is not claculating some values correctly, all “t” amounts should be multiplied by _getRate(), but below amounts are assigned to themselves (or zero):
rStrategicPartnership = s.rStrategicPartnership; rSpinovation = s.rSpinovation;
rUsdtBoost = s.rUsdtBoost;
rHodl = s.rHodl;
rNFT_F = s.rNFT_F; rNFT_L1 = s.rNFT_L1; rNFT_L2 = s.rNFT_L2; rNFT_L3 = s.rNFT_L3;

Suggestion:
calculate all fees correctly:
rStrategicPartnership = s.tStrategicPartnership * currentRate; rSpinovation = s.tSpinovation * currentRate;
rUsdtBoost = s.tUsdtBoost * currentRate;
rHodl = s.tHodl * currentRate;
rNFT_F = s.tNFT_F * currentRate; rNFT_L1 = s.tNFT_L1 * currentRate;
rNFT_L2 = s.tNFT_L2 * currentRate; rNFT_L3 = s.tNFT_L3 * currentRate;

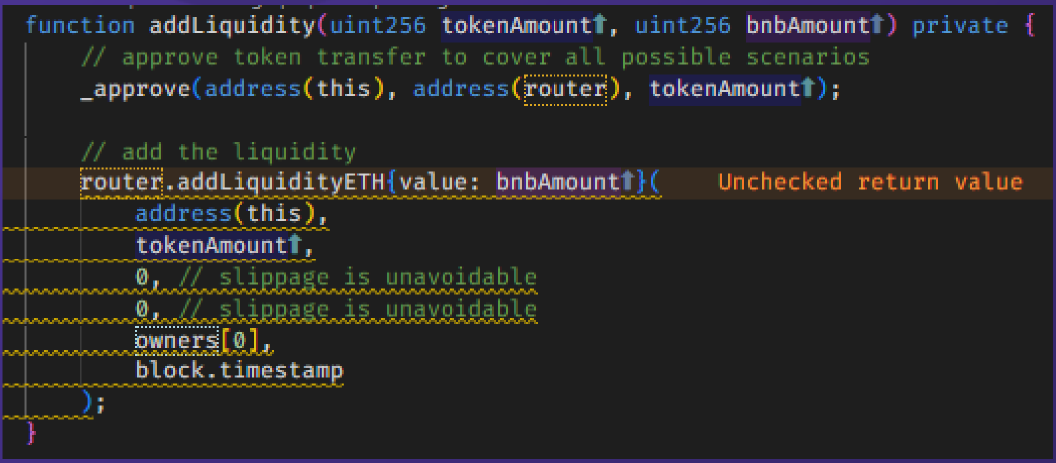
# MEDIUM RISK FINDING

## Auto Liqudity externally owned acocunt

**Severity : Medium**

### Overview

The **addLiquidity** function is a private function responsible for adding liquidity to a decentralized exchange (DEX) pool. It takes **tokenAmount** and **bnbAmount** as parameters, approves the token transfer from the contract to the DEX router, and then adds liquidity using the **router.addLiquidityETH** function.



```
function addLiquidity(uint256 tokenAmount, uint256 bnbAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(router), tokenAmount);

    // add the liquidity
    router.addLiquidityETH{value: bnbAmount}(     Unchecked return value
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        owners[0],
        block.timestamp
    );
}
```

### Recommendation

Instead of relying on an externally owned account for liquidity addition, it is recommended to implement an automated mechanism within the contract itself. This allows for more control, flexibility, and security in managing liquidity.

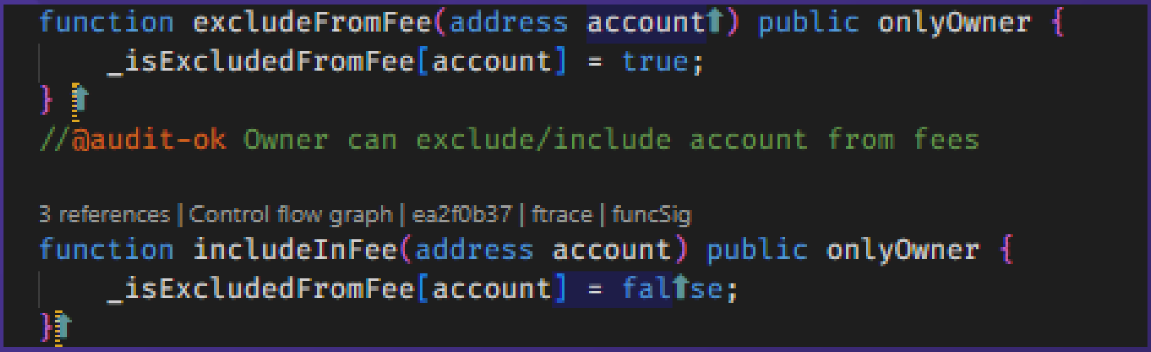
# LOW RISK FINDING

**Owner can exclude account from fees**

## Severity : Low

**Overview**

Excludes/Includes an address from the collection of fees



```
function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}
//@audit-ok Owner can exclude/include account from fees

3 references | Control flow graph | ea2f0b37 | ftrace | funcSig
function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}
```

**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.

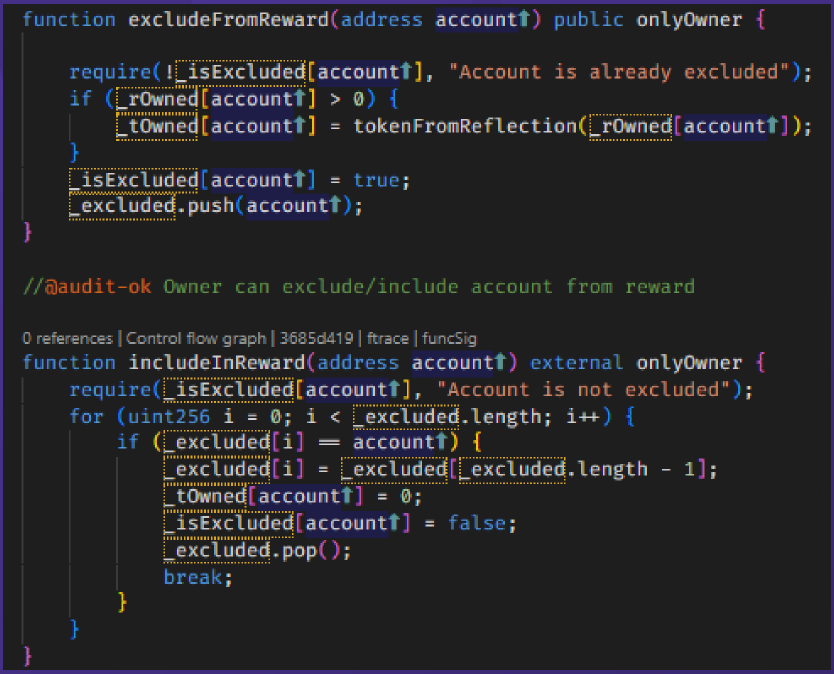
# LOW RISK FINDING

## Owner can exclude/include account from reward

### Severity : Low

### Overview

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



```
function excludeFromReward(address account) public onlyOwner {

    require(!_isExcluded[account], "Account is already excluded");
    if (_rOwned[account] > 0) {
        _tOwned[account] = tokenFromReflection(_rOwned[account]);
    }
    _isExcluded[account] = true;
    _excluded.push(account);
}

//@audit-ok Owner can exclude/include account from reward

0 references | Control flow graph | 3685d419 | ftrace | funcSig
function includeInReward(address account) external onlyOwner {
    require(_isExcluded[account], "Account is not excluded");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

### 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.

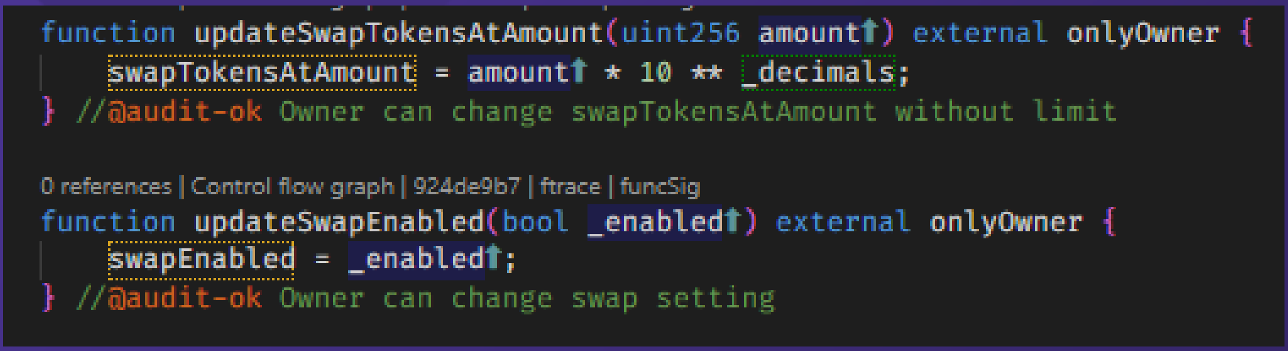
# LOW RISK FINDING

## Owner can change swap 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 updateSwapTokensAtAmount(uint256 amount) external onlyOwner {
    swapTokensAtAmount = amount * 10 ** _decimals;
} //@audit-ok Owner can change swapTokensAtAmount without limit

0 references | Control flow graph | 924de9b7 | ftrace | funcSig
function updateSwapEnabled(bool _enabled) external onlyOwner {
    swapEnabled = _enabled;
} //@audit-ok Owner can change swap setting
```

**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

**Lack of Zero address check**

**Severity : Low**

**Overview**
Functions that allows the owner of the contract to update the buy/sell/transfer fees of the contract. For buy fees and sell fees maximum limit of 5% and transfer fees maximum limit 1%.

**setNFTs**
updateMarketingWallet
updateDevelopmentWallet
updateStrategicPartnershipWallet
updateRouterAndPair

**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.

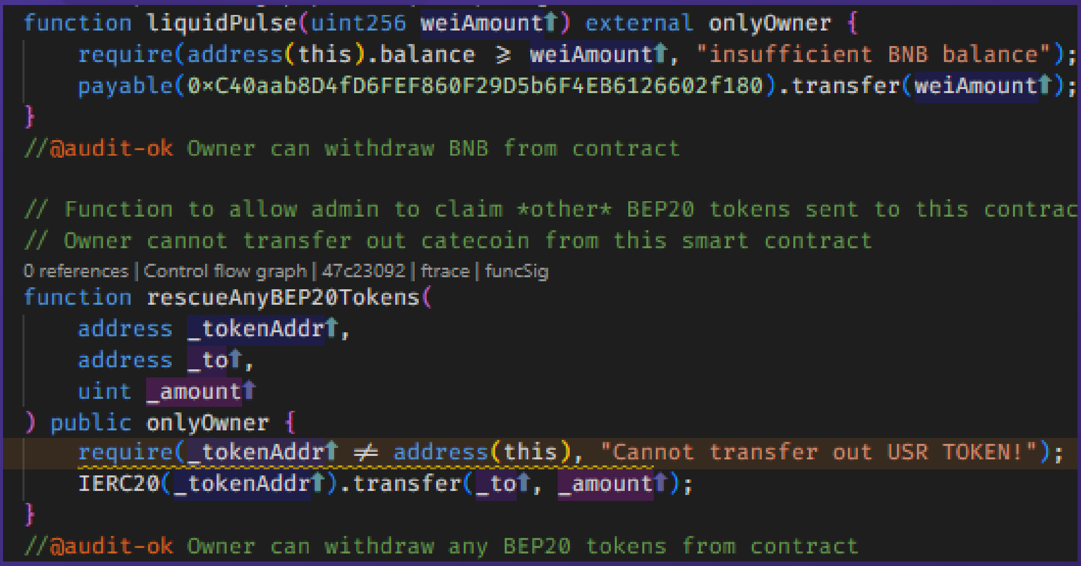
# LOW RISK FINDING

**Owner can claim stuck tokens and BNB**

**Severity : Low**

**Overview**

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



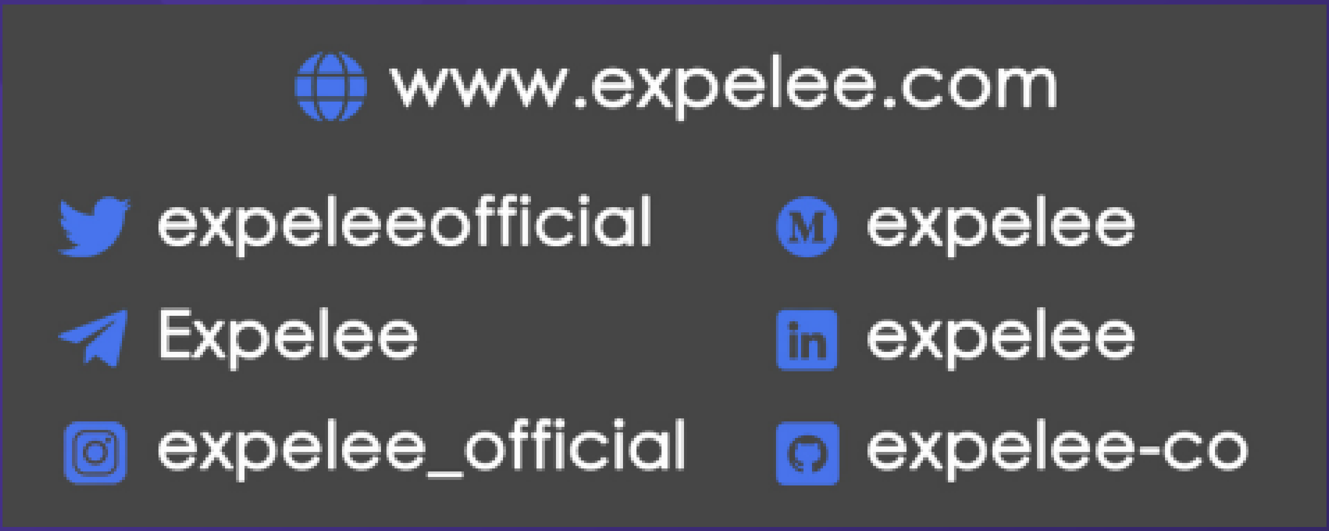
```
function liquidPulse(uint256 weiAmount) external onlyOwner {
    require(address(this).balance >= weiAmount, "insufficient BNB balance");
    payable(0xC40aab8D4fD6FEF860F29D5b6F4EB6126602f180).transfer(weiAmount);
}
//@audit-ok Owner can withdraw BNB from contract

// Function to allow admin to claim *other* BEP20 tokens sent to this contrac
// Owner cannot transfer out catecoin from this smart contract
0 references | Control flow graph | 47c23092 | ftrace | funcSig
function rescueAnyBEP20Tokens(
    address _tokenAddr,
    address _to,
    uint _amount
) public onlyOwner {
    require(_tokenAddr != address(this), "Cannot transfer out USR TOKEN!");
    IERC20(_tokenAddr).transfer(_to, _amount);
}
//@audit-ok Owner can withdraw any BEP20 tokens from contract
```

**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 |

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.

Building the Futuristic Blockchain Ecosystem