expelee

Building the Futuristic Blockchain Ecosystem

SECURITY AUDIT REPORT

SASUKE

TOKEN OVERVIEW

Risk Findings

Severity	Found
🔴 High	1
🟠 Medium	1
🟡 Low	2
🔵 Informational	3

Centralization Risks

Owner Privileges	Description
🟢 Can Owner Set Taxes >25% ?	Not Detected
🔴 Owner Can enable trading ?	Detected
🟢 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	06 Jan, 2024

CONTRACT DETAILS

Token Name: SASUKE

Symbol: Sasuke Inu

Network: BscScan

Decimal: 18

Token Type: BEP - 20

Token Address:
0xcd79b119DD4727fFe50A10CFbcA60a31c37Fd91e

Total Supply: 420,000,000,000,000

Owner's Wallet:
0x8AcE177e642e873270D95cEC55796f91dc4B0267

Deployer's Wallet:
0x4AC8cb73913a9A7e34f82Fac6877af647673210b

CheckSum:
567acbefe2a12642d388659dffd20772

Testnet.
https://testnet.bscscan.com/address/0x989f037bd76d102bc7a4087919767b76ca4ca788#code

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.

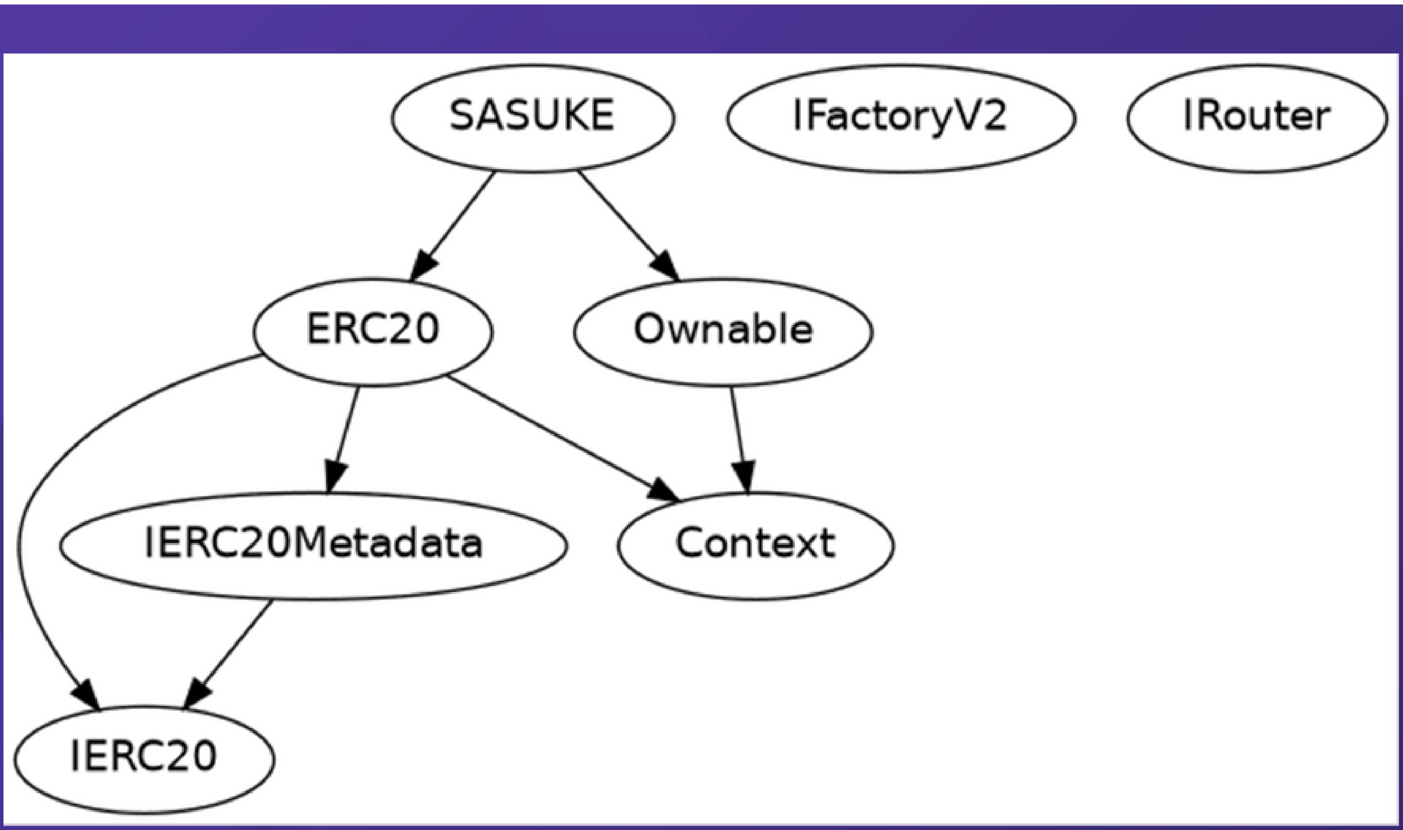
Low Risk

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

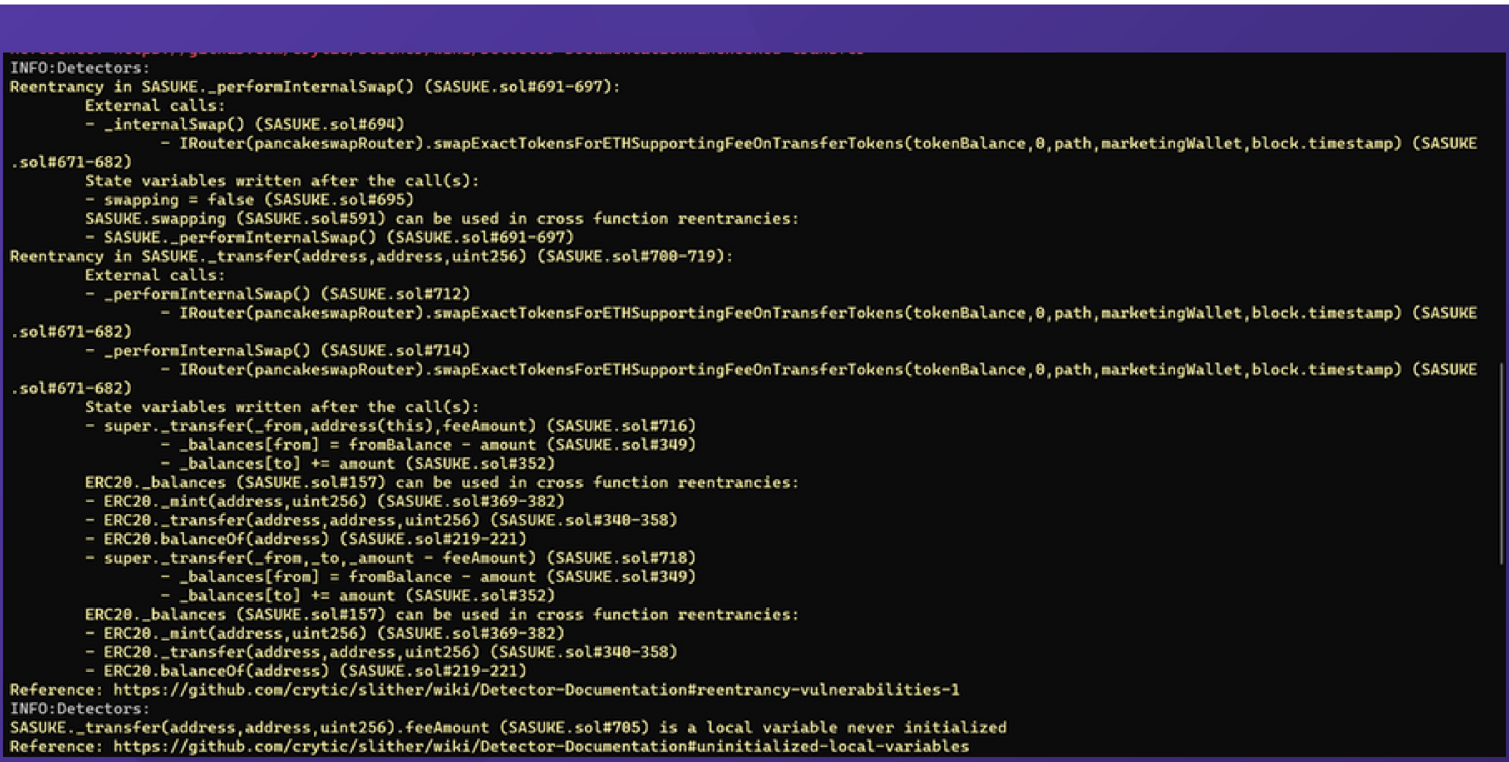
Informational

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

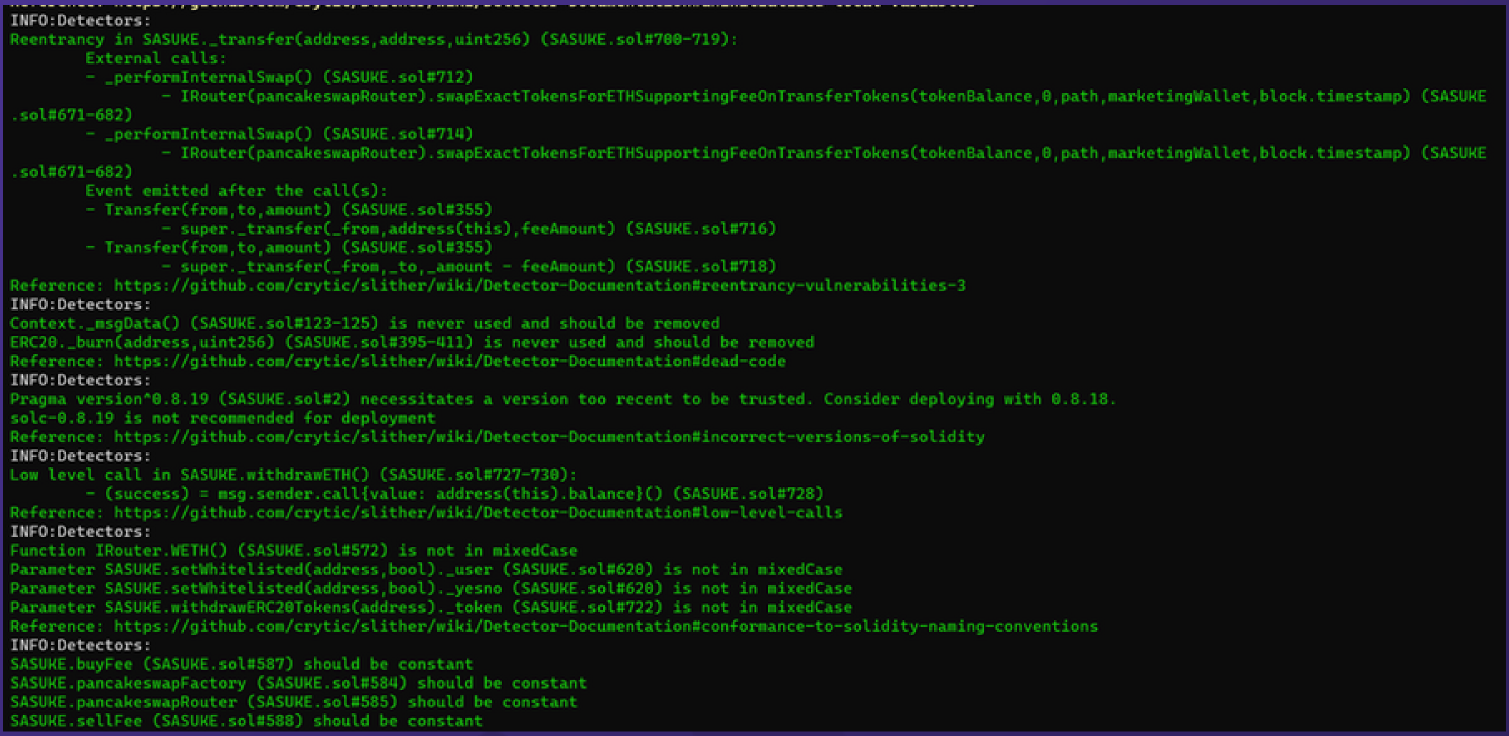
INHERITANCE TREES



STATIC ANALYSIS

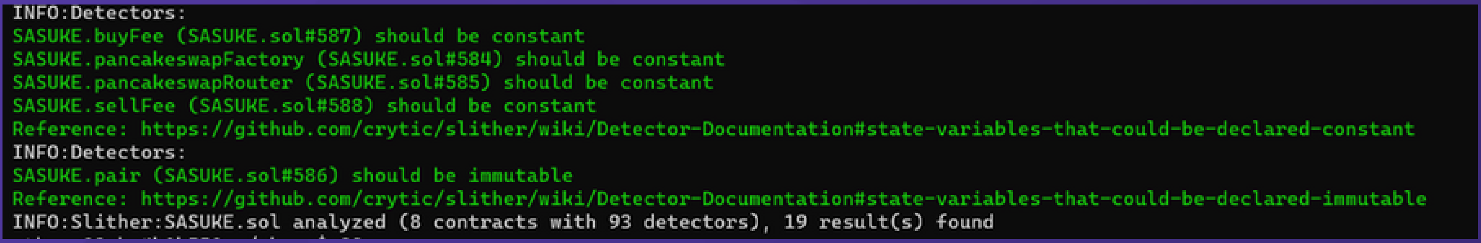


```
INFO:Detectors:
Reentrancy in SASUKE._performInternalSwap() (SASUKE.sol#691-697):
        External calls:
        - _internalSwap() (SASUKE.sol#694)
                - IRouter(pancakeswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (SASUKE
.sol#671-682)
        State variables written after the call(s):
        - swapping = false (SASUKE.sol#695)
        SASUKE.swapping (SASUKE.sol#591) can be used in cross function reentrancies:
        - SASUKE._performInternalSwap() (SASUKE.sol#691-697)
Reentrancy in SASUKE._transfer(address,address,uint256) (SASUKE.sol#700-719):
        External calls:
        - _performInternalSwap() (SASUKE.sol#712)
                - IRouter(pancakeswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (SASUKE
.sol#671-682)
        - _performInternalSwap() (SASUKE.sol#714)
                - IRouter(pancakeswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (SASUKE
.sol#671-682)
        State variables written after the call(s):
        - super._transfer(_from,address(this),feeAmount) (SASUKE.sol#716)
                - _balances[from] = fromBalance - amount (SASUKE.sol#349)
                - _balances[to] += amount (SASUKE.sol#352)
        ERC20._balances (SASUKE.sol#157) can be used in cross function reentrancies:
        - ERC20._mint(address,uint256) (SASUKE.sol#369-382)
        - ERC20._transfer(address,address,uint256) (SASUKE.sol#340-358)
        - ERC20.balanceOf(address) (SASUKE.sol#219-221)
        - super._transfer(_from,_to,_amount - feeAmount) (SASUKE.sol#718)
                - _balances[from] = fromBalance - amount (SASUKE.sol#349)
                - _balances[to] += amount (SASUKE.sol#352)
        ERC20._balances (SASUKE.sol#157) can be used in cross function reentrancies:
        - ERC20._mint(address,uint256) (SASUKE.sol#369-382)
        - ERC20._transfer(address,address,uint256) (SASUKE.sol#340-358)
        - ERC20.balanceOf(address) (SASUKE.sol#219-221)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
SASUKE._transfer(address,address,uint256).feeAmount (SASUKE.sol#705) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```



```
INFO:Detectors:
Reentrancy in SASUKE._transfer(address,address,uint256) (SASUKE.sol#700-719):
        External calls:
        - _performInternalSwap() (SASUKE.sol#712)
                - IRouter(pancakeswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (SASUKE
.sol#671-682)
        - _performInternalSwap() (SASUKE.sol#714)
                - IRouter(pancakeswapRouter).swapExactTokensForETHSupportingFeeOnTransferTokens(tokenBalance,0,path,marketingWallet,block.timestamp) (SASUKE
.sol#671-682)
        Event emitted after the call(s):
        - Transfer(from,to,amount) (SASUKE.sol#355)
                - super._transfer(_from,address(this),feeAmount) (SASUKE.sol#716)
        - Transfer(from,to,amount) (SASUKE.sol#355)
                - super._transfer(_from,_to,_amount - feeAmount) (SASUKE.sol#718)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Context._msgData() (SASUKE.sol#123-125) is never used and should be removed
ERC20._burn(address,uint256) (SASUKE.sol#395-411) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.19 (SASUKE.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in SASUKE.withdrawETH() (SASUKE.sol#727-730):
        - (success) = msg.sender.call{value: address(this).balance}() (SASUKE.sol#728)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IRouter.WETH() (SASUKE.sol#572) is not in mixedCase
Parameter SASUKE.setWhitelisted(address,bool)._user (SASUKE.sol#620) is not in mixedCase
Parameter SASUKE.setWhitelisted(address,bool)._yesno (SASUKE.sol#620) is not in mixedCase
Parameter SASUKE.withdrawERC20Tokens(address)._token (SASUKE.sol#722) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
SASUKE.buyFee (SASUKE.sol#587) should be constant
SASUKE.pancakeswapFactory (SASUKE.sol#584) should be constant
SASUKE.pancakeswapRouter (SASUKE.sol#585) should be constant
SASUKE.sellFee (SASUKE.sol#588) should be constant
```

STATIC ANALYSIS



```
INFO:Detectors:
SASUKE.buyFee (SASUKE.sol#587) should be constant
SASUKE.pancakeswapFactory (SASUKE.sol#584) should be constant
SASUKE.pancakeswapRouter (SASUKE.sol#585) should be constant
SASUKE.sellFee (SASUKE.sol#588) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
SASUKE.pair (SASUKE.sol#586) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
INFO:Slither:SASUKE.sol analyzed (8 contracts with 93 detectors), 19 result(s) found
```

TESTNET VERSION

1- Approve (passed):
https://testnet.bscscan.com/tx/0xf27098245fe8796ac5fe4e58915c91daae14b34c76899fc5e4fbc6691a370f1a

2- Increase Allowance (passed):
https://testnet.bscscan.com/tx/0x80c42db6658fb00b3836affb3c19b6dc47d6aaf9538fd1804be85b5413e9b48f

3- Decrease Allowance (passed):
https://testnet.bscscan.com/tx/0x26002a17275e52c50c78cb47a98401e43bc5beb9a1b88c1b86e37d377886a1cb

4- Enable Trading (passed):
https://testnet.bscscan.com/tx/0x768519cc4667b86cc2e832b3cb71866d0659365b0c0e85180eae683b8a7791d2

5- Update Marketing Wallet (passed):
https://testnet.bscscan.com/tx/0xa8ed63553a032a8a8d77b04c5f06513ddba54ad80ab1f7d679df189c37a86934

MANUAL REVIEW

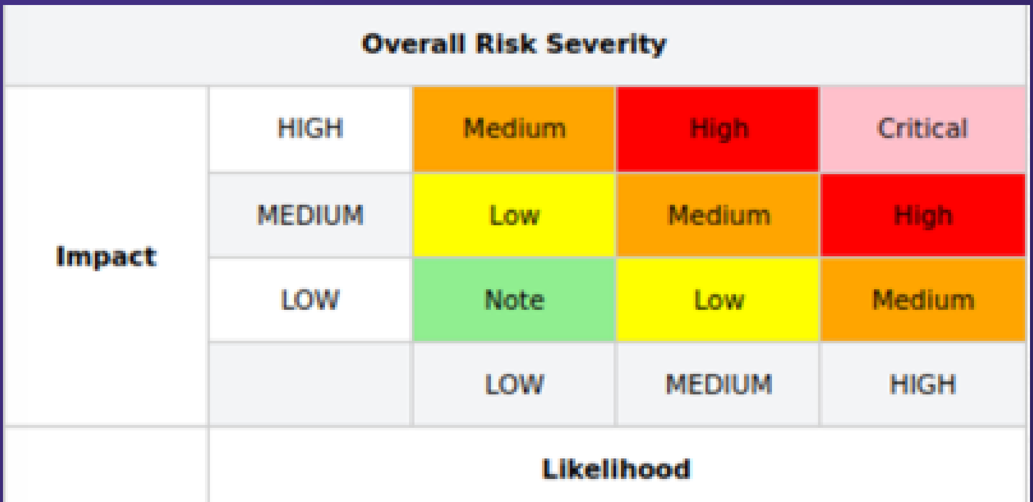
Severity Criteria

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

Enabling Trades

Category: Centralization
Severity: High
Function: EnableTrading
Status:Open

Overview:
The EnableTrading function permits only the contract owner to activate trading capabilities. Until this function is executed, no investors can buy, sell, or transfer their tokens. This places a high degree of control and centralization in the hands of the contract owner.

```
function enableTrading() public onlyOwner {
require(!tradingEnabled, "Already enabled");
   tradingEnabled = true;
emit TradingEnabled(block.timestamp);
 }
```

Suggestion:

To reduce centralization and potential manipulation, consider one of the following approaches:

1. Automatically enable trading after a specified condition, such as the completion of a presale, is met.

HIGH RISK FINDING

2. If manual activation is still desired, consider transferring the ownership of the contract to a trustworthy, third-party entity like a certified "PinkSale Safu" developer. This can give investors more confidence in the eventual activation of trading capabilities, mitigating concerns of potential bad-faith actions by the original owner.

MEDIUM RISK FINDING

Missing Require Check.

Category: **Centralization**
Severity: **Medium**
Function: updateMarketingWallet
Status:Open

Overview:
The owner can set any arbitrary address excluding zero address as this is not recommended because if the owner will set the address to the contract address, then the Eth will not be sent to that address and the transaction will fail and this will lead to a potential honeypot in the contract.

```
  function updateMarketingWallet(address marketing_)
public onlyOwner {
require(marketing_ != address(0), "address zero not
accepted");
   marketingWallet = marketing_;
emit MarketinWalletUpdated(marketing_);

 }
```

Suggestion:
It is recommended that the address should not be able to be set as a contract address.

LOW RISK FINDING

Missing Events

Category: Centralization
Severity: Low
Subject: Missing Events
Status:Open

Overview:
They serve as a mechanism for emitting and recording data onto the blockchain, making it transparent and easily accessible.

```
  function setWhitelisted(address _user, bool _yesno) public
onlyOwner {
    whitelist[_user] = _yesno;
  }
```

LOW RISK FINDING

Missing Visibility

Severity: Low
Subject: Visibility
Status:Open

Overview: It's simply saying that no visibility was specified, so it's going with the default. This has been related to security issues in contracts.

bool swapping.

Suggestion:
You can easily silence the warning by adding the public/private.

INFORMATIONAL RISK FINDING

Severity: Informational
Subject: Floating Pragma.
Status: Open

Overview: It is considered best practice to pick one compiler version and stick with it. With a floating pragma, contracts may accidentally be deployed using an outdated.

```
pragma solidity ^0.8.13;
```

Suggestion:
Adding the latest constant version of solidity is recommended, as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs.

INFORMATIONAL RISK FINDING

Severity: Informational
Subject: uint256
Status: Open

Overview: Use uit256 instead of uint. uint is an alias for uint256 and is not recommended for use. The variable size should be clarified, as this can cause issues when encoding data with selectors if the alias is mistakenly used within the signature string.

```
uint public buyFee = 3;
uint public sellFee = 3;
uint public minimumForInternalSwap;
```

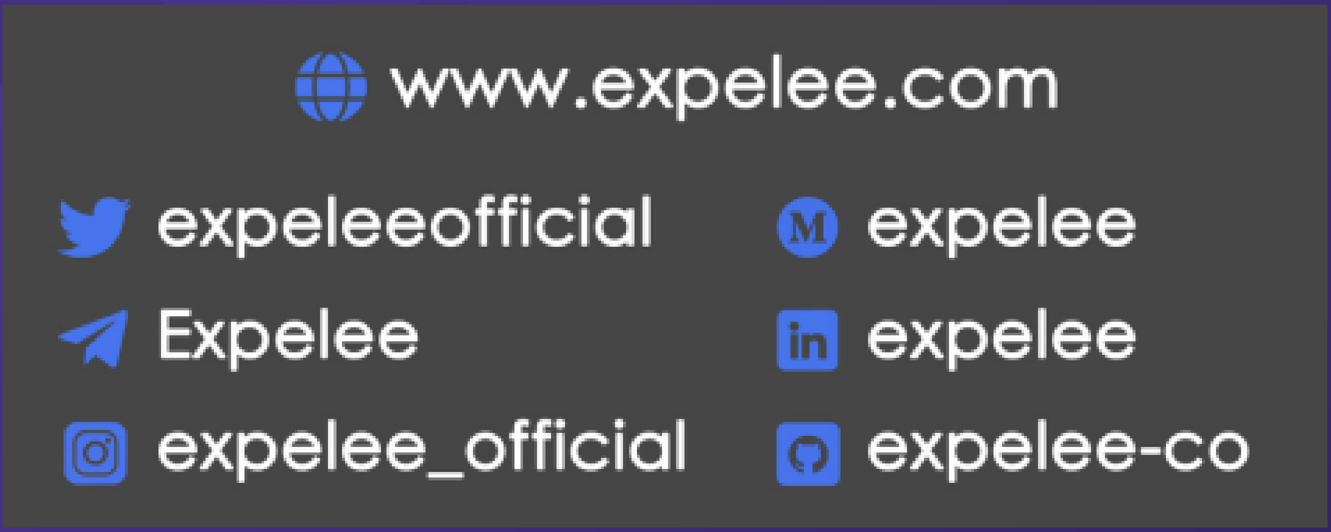
INFORMATIONAL RISK FINDING

Severity: Optimization
Subject: Remove unused code.
Status: Open
Overview:
Unused variables are allowed in Solidity, and they do. not pose a direct security issue. It is the best practice. though to avoid them.

```
function _msgData() internal view virtual returns (bytes
calldata) {
return msg.data;
 }
```

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.

expelee

Building the Futuristic Blockchain Ecosystem