auditace.tech

AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

PEPEGROK

DATED : 14 March, 2024

MANUAL TESTING

Centralization – Owner Can Mint Tokens

Severity: High

Function: mint

Status: Open

Overview:

The owner is able to mint unlimited tokens which is not recommended as this functionality can cause the token to lose its value and the owner can also use it to manipulate the price of the token.

```
function mint(uint256 amount) public onlyOwner returns (bool) {
        _mint(_msgSender(), amount);
        return true;
    }
```

Suggestion:

It is recommended that the total supply of the token should not be changed after initial deployment.

MANUAL TESTING

Centralization – Missing Require Check

Severity: High

Function: Set Marketing Pool

Status: Open

Overview:

The owner can set any arbitrary address excluding zero address as this is not recommended because if the owner sets 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 setMarketingPool(address _marketingPool) external onlyOwner {
        require(_marketingPool != address(0), "Marketing Wallet cannot be zero ad-
dress");
        marketingPool = _marketingPool;
        excludedFromFees[_marketingPool] = true;
        emit ChangeMarketingPool(_marketingPool);
    }
```

Suggestion:

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

AUDIT SUMMARY

Project name – PEPEGROK

Date: 14 March, 2024

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: **Major Flag High Risk**

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	2	0	3	2
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1- Manual Review:
A line by line code review has been performed by audit ace team.

2- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.

3- Slither :
The code has undergone static analysis using Slither.

Testnet version:
The tests were performed using the contract deployed on the BSC Testnet, which can be found at the following address:
https://testnet.bscscan.com/address/0x00e895a8ea1768b10b21bb8754cb0ea3fe397b02#code

Token Information

Token Name : PEPEGROK

Token Symbol: PEPEGROK

Decimals: 18

Token Supply: 1000000000

Network: BscScan

Token Type: BEP-20

Token Address:
0xDe38d1E43cca3D7CE0ad853889B6069Cac3A5AE7

Checksum:
A2032c616934aeb47e6039f76b20d213

Owner:
0xbaC14f1D89f03Ad7E78a93eBCBc6AD13aEdB4b2F
(at time of writing the audit)

Deployer:
0xbaC14f1D89f03Ad7E78a93eBCBc6AD13aEdB4b2F

TOKEN OVERVIEW

Fees:

Buy Fee: 2-5%

Sell Fee: 2-5%

Transfer Fee: 0%

Fees Privilege: Owner

Ownership: Owned

Minting: Yes

Max Tx Amount/ Max Wallet Amount: No

Blacklist: No

AUDIT METHODOLOGY

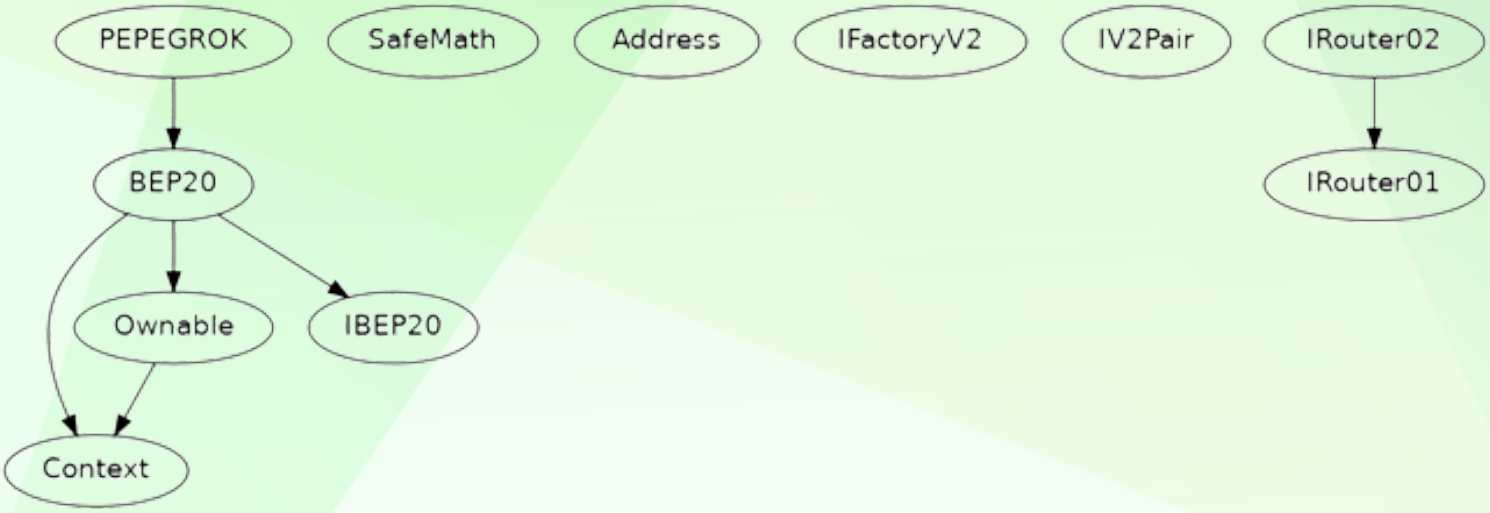
The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code isexercised when we run the test cases.
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

VULNERABILITY CHECKLIST

- [x] Return values of low-level calls
- [x] Private modifier
- [x] Multiple Sends
- [x] Using Suicide
- [x] Gas Limitand Loops
- [x] Address hardcoded
- [x] Exception Disorder
- [x] Using inline assembly
- [x] Divide before multiply
- [x] Missing Zero Address Validation
- [x] Compiler version not fixed
- [x] **Gasless Send**
- [x] Using block.timestamp
- [x] Re-entrancy
- [x] Tautology or contradiction
- [x] Timestamp Dependence
- [x] Revert/require functions
- [x] Use of tx.origin
- [x] Integer overflow/underflow
- [x] Dangerous strict equalities
- [x] Using SHA3
- [x] Using throw

INHERITANCE TREE



STATIC ANALYSIS

A static analysis of the code was performed using Slither.

No issues were found.

```
INFO:Detectors:
BEP20.constructor(string,string).name (PEPEGROK.sol#284) shadows:
        - BEP20.name() (PEPEGROK.sol#294-296) (function)
        - IBEP20.name() (PEPEGROK.sol#62) (function)
BEP20.constructor(string,string).symbol (PEPEGROK.sol#284) shadows:
        - BEP20.symbol() (PEPEGROK.sol#298-300) (function)
        - IBEP20.symbol() (PEPEGROK.sol#60) (function)
BEP20.allowance(address,address).owner (PEPEGROK.sol#319) shadows:
        - Ownable.owner() (PEPEGROK.sol#31-33) (function)
BEP20._approve(address,address,uint256).owner (PEPEGROK.sol#376) shadows:
        - Ownable.owner() (PEPEGROK.sol#31-33) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
PEPEGROK.setPair(address)._pair (PEPEGROK.sol#422) lacks a zero-check on :
                - pair = _pair (PEPEGROK.sol#423)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in PEPEGROK.withdrawStuckTokens(address,address) (PEPEGROK.sol#477-481):
        External calls:
        - _sent = IBEP20(_token).transfer(_to,_contractBalance) (PEPEGROK.sol#479)
        Event emitted after the call(s):
        - TransferForeignToken(_token,_contractBalance) (PEPEGROK.sol#480)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Address.isContract(address) (PEPEGROK.sol#132-139) uses assembly
        - INLINE ASM (PEPEGROK.sol#135-137)
Address._functionCallWithValue(address,bytes,uint256,string) (PEPEGROK.sol#177-197) uses assembly
        - INLINE ASM (PEPEGROK.sol#189-192)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
PEPEGROK._transfer(address,address,uint256) (PEPEGROK.sol#442-462) compares to a boolean constant:
        -liquidityPool[sender] == true (PEPEGROK.sol#452)
PEPEGROK._transfer(address,address,uint256) (PEPEGROK.sol#442-462) compares to a boolean constant:
        -liquidityPool[recipient] == true (PEPEGROK.sol#454)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

```
INFO:Detectors:
Function IRouter01.WETH() (PEPEGROK.sol#214) is not in mixedCase
Parameter PEPEGROK.setPair(address)._pair (PEPEGROK.sol#422) is not in mixedCase
Parameter PEPEGROK.setRouter(address)._router (PEPEGROK.sol#426) is not in mixedCase
Parameter PEPEGROK.setMarketingPool(address)._marketingPool (PEPEGROK.sol#430) is not in mixedCase
Parameter PEPEGROK.setLiquidityPoolStatus(address,bool)._lpAddress (PEPEGROK.sol#437) is not in mixedCase
Parameter PEPEGROK.setLiquidityPoolStatus(address,bool)._status (PEPEGROK.sol#437) is not in mixedCase
Parameter PEPEGROK.setTaxes(uint8,uint8)._sellTax (PEPEGROK.sol#464) is not in mixedCase
Parameter PEPEGROK.setTaxes(uint8,uint8)._buyTax (PEPEGROK.sol#464) is not in mixedCase
Parameter PEPEGROK.setExcludedFromFees(address,bool)._address (PEPEGROK.sol#472) is not in mixedCase
Parameter PEPEGROK.withdrawStuckTokens(address,address)._token (PEPEGROK.sol#477) is not in mixedCase
Parameter PEPEGROK.withdrawStuckTokens(address,address)._to (PEPEGROK.sol#477) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (PEPEGROK.sol#14)" inContext (PEPEGROK.sol#8-17)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Reentrancy in PEPEGROK.clearStuckEthers(uint256) (PEPEGROK.sol#483-487):
        External calls:
        - address(msg.sender).transfer((amountETH * amountPercentage) / 100) (PEPEGROK.sol#485)
        Event emitted after the call(s):
        - StuckEthersCleared() (PEPEGROK.sol#486)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
Variable IRouter01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (PEPEGROK.sol#226) is too similar to IRouter
01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (PEPEGROK.sol#227)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Detectors:
PEPEGROK.constructor() (PEPEGROK.sol#409-420) uses literals with too many digits:
        - _mint(_msgSender(),1000000000 * 10000000000000000000) (PEPEGROK.sol#418)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
PEPEGROK.deadAddress (PEPEGROK.sol#398) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Slither:PEPEGROK.sol analyzed (11 contracts with 93 detectors), 52 result(s) found
```

FUNCTIONAL TESTING

1- Approve (passed):
https://testnet.bscscan.com/tx/0x0872f330a45f61e5088b831b092b2dd0cd71d580733a0e62fc4512fd9d01301c

2- Set Excluded from Fees (passed):
https://testnet.bscscan.com/tx/0x0322a11d1e1dd881c6b578524c7c2dddc305381dd0832c3d2f561e3bed5b3807

3- Mint (passed):
https://testnet.bscscan.com/tx/0x25117694d4c8c18c2dfe158f6c2d01495575598e95c4f8960f9e1e82d8f08c29

4- Set Marketing Pool (passed):
https://testnet.bscscan.com/tx/0xf9e54d6dd505acf6c4a626230f48bc7387acebacf4982f542d0b10ccc18a0d85

5- Set Taxes (passed):
https://testnet.bscscan.com/tx/0x93354c3c413c4424031a457ebaa356ef9407d1417b3745fd487ff7681315f6c6

POINTS TO NOTE

- The owner can transfer ownership.
- The owner can renounce ownership.
- The owner can mint token.
- The owner can set pair address.
- The owner can set the Router address.
- The owner can set the marketing pool address.
- The owner can set liquidity pool status.
- The owner can set buy/sell tax to not more than 5%.
- The owner can exclude address from fees.
- The owner can rescue ETH.

CLASSIFICATION OF RISK

Severity	Description
Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
Gas Optimization /Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
Critical	0
High-Risk	2
Medium-Risk	0
Low-Risk	3
Gas Optimization / Suggestions	2

MANUAL TESTING

Centralization – Owner Can Mint Tokens

Severity: High

Function: mint

Status: Open

Overview:

The owner is able to mint unlimited tokens which is not recommended as this functionality can cause the token to lose its value and the owner can also use it to manipulate the price of the token.

```
function mint(uint256 amount) public onlyOwner returns (bool) {
        _mint(_msgSender(), amount);
        return true;
    }
```

Suggestion:

It is recommended that the total supply of the token should not be changed after initial deployment.

MANUAL TESTING

Centralization – Missing Require Check

Severity: High

Function: Set Marketing Pool

Status: Open

Overview:

The owner can set any arbitrary address excluding zero address as this is not recommended because if the owner sets 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 setMarketingPool(address _marketingPool) external onlyOwner {
        require(_marketingPool != address(0), "Marketing Wallet cannot be zero ad-
dress");
        marketingPool = _marketingPool;
        excludedFromFees[_marketingPool] = true;
        emit ChangeMarketingPool(_marketingPool);
    }
```

Suggestion:

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

MANUAL TESTING

Centralization – Local Variable Shadowing

Severity: Low

Function: _approve and allowance

Status: Open

Overview:

```
function allowance(address owner, address spender) public override view returns
(uint256) {
        return _allowances[owner][spender];
    }
function _approve (address owner, address spender, uint256 amount) internal {
        require(owner != address(0), 'BEP20: approve from the zero address');
        require(spender != address(0), 'BEP20: approve to the zero address');
        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

Suggestion:

Rename the local variable that shadows another component.

MANUAL TESTING

Centralization – Missing Zero Address

Severity: Low

Function: Zero Check

Status: Open

Overview:

Functions can take a zero address as a parameter (0x00000...). If a function parameter of address type is not properly validated by checking for zero addresses, there could be serious consequences for the contract's functionality.

```
function setPair(address _pair) external onlyOwner{
        pair = _pair;
    }
```

MANUAL TESTING

Optimization

Severity: Low

Subject: Old Pragma Solidity version

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 0.6.12;
```

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.

MANUAL TESTING

Optimization

Severity: Informational

Function: Remove Safe Math

Status: Open

Line: 83-128

Overview:

compiler version above 0.8.0 can control arithmetic overflow/underflow, it is recommended to remove the unwanted code to avoid high gas fees.

MANUAL TESTING

Optimization

Severity: Optimization

Function: 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 memory) {
        this;
        return msg.data;
    }
interface IV2Pair {
    function factory() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 re-
serve1, uint32 blockTimestampLast);
    function sync() external;
}
function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value
    ) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, 'Address: low-level call
with value failed');
    }
 function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, 'Address: insufficient balance');
        (bool success, ) = recipient.call{value: amount}('');
        require(success, 'Address: unable to send value, recipient may have revert-
ed');
    }
function functionCall(address target, bytes memory data) internal returns (bytes
memory) {
        return functionCall(target, data, 'Address: low-level call failed');
```

Suggestion:

To reduce high gas fees. It is suggested to remove unused code from the contract.

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 guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace 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 Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always 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 Auditace team disclaims any liability for the resulting losses.

ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.

https://auditace.tech/

https://t.me/Audit_Ace

https://twitter.com/auditace_

https://github.com/Audit-Ace