auditace.tech

AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

PepuLock

DATED : 27 August 2025

AUDIT SUMMARY

Project name – PepuLock

Date: 27 August 2025

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

Audit Status: **Passed with medium risk.**

Issues Found

Status	Critical	High	Medium	Low	Informational
Open	0	0	1	0	0
Acknowledged	0	0	0	0	0
Resolved	0	0	1	1	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.

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

Token Information

Name: PepuLock

Symbol: -

Decimals: -

Network: -

Type: ETH L2 network

Owner: -

Deployer: -

Token Supply: -

Checksum: afa6dc77b1f71cf281fa48605b74e722

TOKEN OVERVIEW

Fees: -

Fee Privilege: Owner

Ownership: Owned

Minting: No

Max Tx: 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

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

POINTS TO NOTE

- The owner can update the fees by more than 100%.
- The owner can update the fee wallet address.
- The validlock address can unlock and edit the lock.
- The validlock address can edit the lock description.

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	0
Medium-Risk	1
Low-Risk	0
Optimization/ Informational	0

MANUAL TESTING

Centralization –The owner can set high fees

Severity: Medium

Function: updateFee

Status: Open

Overview:

The updateFee function allows the contract owner to set arbitrarily high fees without limits or time delays. This could potentially trap user funds by making the contract economically unusable through excessive fees, posing a significant centralization risk.

```
function updateFee(uint256 newFee) external onlyOwner {
 fee = newFee;
 emit FeeUpdated(newFee);
 }
```

Suggestions:

Implement a maximum fee cap of 25% by adding a constant and require statement: require(newFee<= MAX_FEE_BPS, "Fee exceeds 25%”);

MANUAL TESTING

Centralization –Insufficient Validation for Fee Wallet Address(Potential Honeypot).

Severity: Medium

Status: Fixed

Overview:

While the function checks for a zero address, it doesn't validate if the new wallet address is a
contract that may not properly handle ETH transfers. If the fee wallet is set to a contract
address that cannot receive ETH, it could block all future token locking operations.

```
function updateFeeWallet(address newWallet) external onlyOwner {
 if (newWallet == address(0)) revert InvalidFeeWallet();
 feeWallet = newWallet;
 emit FeeWalletUpdated(newWallet);
 }
```

Suggestions:
Add validation for contract addresses and ensure they can receive ETH.

MANUAL TESTING

Centralization - Lack of Two-Step Transfer Pattern in Critical Address Updates

Severity: Low

Status: Fixed

Overview:

The contract implements critical address changes (like fee wallet updates) in a single step without verification. This common smart contract vulnerability risks permanent loss of funds or protocol functionality if addresses are incorrectly set through human error, typos, or compromised admin keys.

Suggestions:

Implement a two-step transfer pattern requiring explicit acceptance from new addresses before finalizing any critical address changes.

General Impact:

1- Funds could be permanently lost
2- Protocol functionality could be broken
3- No recovery mechanism available
4- Affects all future fee collections
5- Compromised admin keys could cause immediate damage

This is a common security pattern issue that affects many protocols and should be addressed using well-established two-step ownership/role transfer patterns from trusted libraries like OpenZeppelin.

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