



Advanced Manual Smart Contract Audit

May 5, 2025

 [CoinsultAudits](#)

 t.me/coinsult_tg

 coinsult.net

Audit requested by



Pepe ETH Ico

No contract address

Global Overview

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
Informational	0	0	0	0
Low-Risk	0	0	0	0
Medium-Risk	1	0	1	0
High-Risk	1	0	1	0

Table of Contents

1. Audit Summary

- 1.1 Audit scope
- 1.2 Tokenomics
- 1.3 Source Code

2. Disclaimer

3. Global Overview

- 3.1 Informational issues
- 3.2 Low-risk issues
- 3.3 Medium-risk issues
- 3.4 High-risk issues

4. Vulnerabilities Findings

5. Contract Privileges

- 5.1 Maximum Fee Limit Check
- 5.2 Contract Pausability Check
- 5.3 Max Transaction Amount Check
- 5.4 Exclude From Fees Check
- 5.5 Ability to Mint Check
- 5.6 Ability to Blacklist Check
- 5.7 Owner Privileges Check

6. Notes

- 6.1 Notes by Coinsult
- 6.2 Notes by Pepe ETH Ico

7. Contract Snapshot

8. Website Review

9. Certificate of Proof

Audit Summary

Project Name	Pepe ETH Ico
Website	
Blockchain	-
Smart Contract Language	Solidity
Contract Address	
Audit Method	Static Analysis, Manual Review
Date of Audit	5 May 2025

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Audit Scope

Coinsult was commissioned by Pepe ETH Ico to perform an audit based on the following code:

<https://github.com/FUTUREPEPE/ico-contract/blob/main/pepeEthIco.sol>

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

Used tools

- Slither: Solidity static analysis framework
- Remix: IDE Developer Tool
- CWE: Common Weakness Enumeration
- SWC: Smart Contract Weakness Classification and Test Cases
- DEX: Testnet Blockchains

Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

Vulnerability Level	Description
● Informational	Does not compromise the functionality of the contract in any way
● Low-Risk	Won't cause any problems, but can be adjusted for improvement
● Medium-Risk	Will likely cause problems and it is recommended to adjust
● High-Risk	Will definitely cause problems, this needs to be adjusted

Coinsult has four statuses that are used for each risk level. Below we explain them briefly.

Risk Status	Description
Total	Total amount of issues within this category
Pending	Risks that have yet to be addressed by the team
Acknowledged	The team is aware of the risks but does not resolve them
Resolved	The team has resolved and remedied the risk

SWC Attack Analysis

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Description	Status
SWC-100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed
SWC-107	Reentrancy	Passed
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed

SWC-116	Block values as a proxy for time	Passed
SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Griefing	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Typographical Error	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed
SWC-131	Presence of unused variables	Passed
SWC-132	Unexpected Ether balance	Passed
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Passed
SWC-134	Message call with hardcoded gas amount	Passed
SWC-135	Code With No Effects	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed

Error Code	Description
CSM-01	Self Referral

● **Medium-Risk:** Should be fixed, could bring problems.

Self Referral

```
function buyWithNative(address referrer) external payable notPaused nonReentrant {
    require(msg.value > 0, "Must send ETH to buy tokens");
    uint256 adminAmount = msg.value;
    uint256 tokenAmount = getTokenFromNative(adminAmount);
    require(token.balanceOf(address(this)) >= tokenAmount, "Not enough tokens in contract");
    _checkSupply(tokenAmount);
    if(isReferral[referrer]) {
        uint256 commissionAmount = (msg.value * commissions[referrer]) / 100;
        adminAmount = msg.value - commissionAmount;
        Address.sendValue(payable(referrer), commissionAmount);
        uint256 usdAmount = getUSDValue(commissionAmount);
        commissionCollect[referrer].commissionUSD += usdAmount;
        commissionCollect[referrer].totalTokenSale += tokenAmount;
    }
    userDeposits[msg.sender].push(Purchase({amount: msg.value, currency: "ETH", tokenAmount: tokenAmount}));
    ethRaised += msg.value;
    totalTokenSold += tokenAmount;
    Address.sendValue(payable(adminWallet), adminAmount);
    token.safeTransfer(msg.sender, tokenAmount);
    emit TokensPurchased(msg.sender, msg.value, tokenAmount);
}
```

Recommendation

Both buyWithNative and buyWithUSDT accept an arbitrary referrer address, check only that isReferral[referrer] == true, and then:

Compute tokens based on the full payment amount.

Pay the commission out (in ETH or USDT) to the referrer, then forward the remainder to adminWallet.


Issue tokens for the full amount, ignoring that a cut was taken as commission.


If a registered influencer simply passes their own address as referrer, they effectively:

Pay 100% of funds in;

Receive back their commission (e.g. 10% of the ETH) immediately;

Still get tokens priced as if they'd paid the full amount.

Error Code	Description
CSH-01	Decimals Logic  Acknowledged

 **High-Risk:** Must be fixed, will bring problems.

Decimals Logic Acknowledged

```
commissionCollect[referrer].commissionUSD += usdAmount;  
commissionCollect[referrer].totalTokenSale += tokenAmount;
```

Recommendation

ETH path: `commissionCollect[referrer].commissionUSD += getUSDValue(commissionAmount)` → stored in USD with 8 decimals.

USDT path: `commissionCollect[referrer].commissionUSD += commissionAmount` → stored in USDT's 6-decimals units.

I was wrote the same in comment of `getUSDValue` function but this function returning the value in 6 decimals, as you can see I was doing the conversion in the same function so at the end I am storing the USDT value in 6 decimals and I already tested it out. `uint256 usdAmount = getUSDValue(commissionAmount); commissionCollect[referrer].commissionUSD += usdAmount;`

Other Owner Privileges Check

Error Code	Description
CEN-100	Centralization: Operator Privileges

Coinsult lists all important contract methods which the owner can interact with.

- **Full pause / unpause** via `toggleSale()`—owner can halt or resume at will.
- **Stage-timing overrides** (`setCustomStageStartTime`) and **fixed-price switches** (`setUseFixedPrice`) give the owner unilateral control over pricing and schedule.
- **Commission rates** and the list of “influencers” are owner-managed.
- **Fund withdrawal:** owner can drain **all** ETH (`withdrawNative`) or **any** ERC20 (`withdrawToken`) to `adminWallet` at any time, even before the ICO cap is reached.

Takeaway: if the owner key is compromised, an attacker can stop sales, change prices, and steal both incoming funds and the tokens intended for buyers.

Addresses access to `buyWithCard` function can “buy” tokens for free.

Note from the team:

These controls were intentionally implemented to allow necessary operational flexibility, such as responding to market conditions, addressing edge cases, and ensuring smooth sale execution. However, we emphasize that this is a temporary measure—ownership will be permanently renounced immediately after the ICO concludes, transitioning the contract to a fully decentralized and immutable state.

Additionally, we take security seriously and will implement strict safeguards for the owner wallet, including multi-signature controls. Ex: `safeWallet`

Notes

Notes by Pepe ETH Ico

These controls were intentionally implemented to allow necessary operational flexibility, such as responding to market conditions, addressing edge cases, and ensuring smooth sale execution. However, we emphasize that this is a temporary measure—ownership will be permanently renounced immediately after the ICO concludes, transitioning the contract to a fully decentralized and immutable state.

Additionally, we take security seriously and will implement strict safeguards for the owner wallet, including multi-signature controls. Ex: `safeWallet`

Notes by Coinsult

- We assume Chainlink Price Aggregation works as a black box.
- **usdtRaised includes referral cuts**
`usdtRaised += usdtAmount;` counts the full USDT spent, even the slice immediately forwarded to influencers. If you mean “net USDT raised,” you should only add the post-commission amount.
- **Return of dynamic string currency in `getUserTxData`**
Storing a string in a struct and returning it can be awkward in many SDKs. An enum or uint8 code would be more gas- and ABI-friendly.
- **Unused pragma declarations**
The file has three repeated `pragma solidity 0.8.20;` lines. Clean up to one to improve readability.

Contract Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.20;

abstract contract ReentrancyGuard {
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    constructor() {
        _status = _NOT_ENTERED;
    }
    modifier nonReentrant() {
        _nonReentrantBefore();
        _;
        _nonReentrantAfter();
    }

    function _nonReentrantBefore() private {
        // On the first call to nonReentrant, _status will be _NOT_ENTERED
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

        // Any calls to nonReentrant after this point will fail
        _status = _ENTERED;
    }

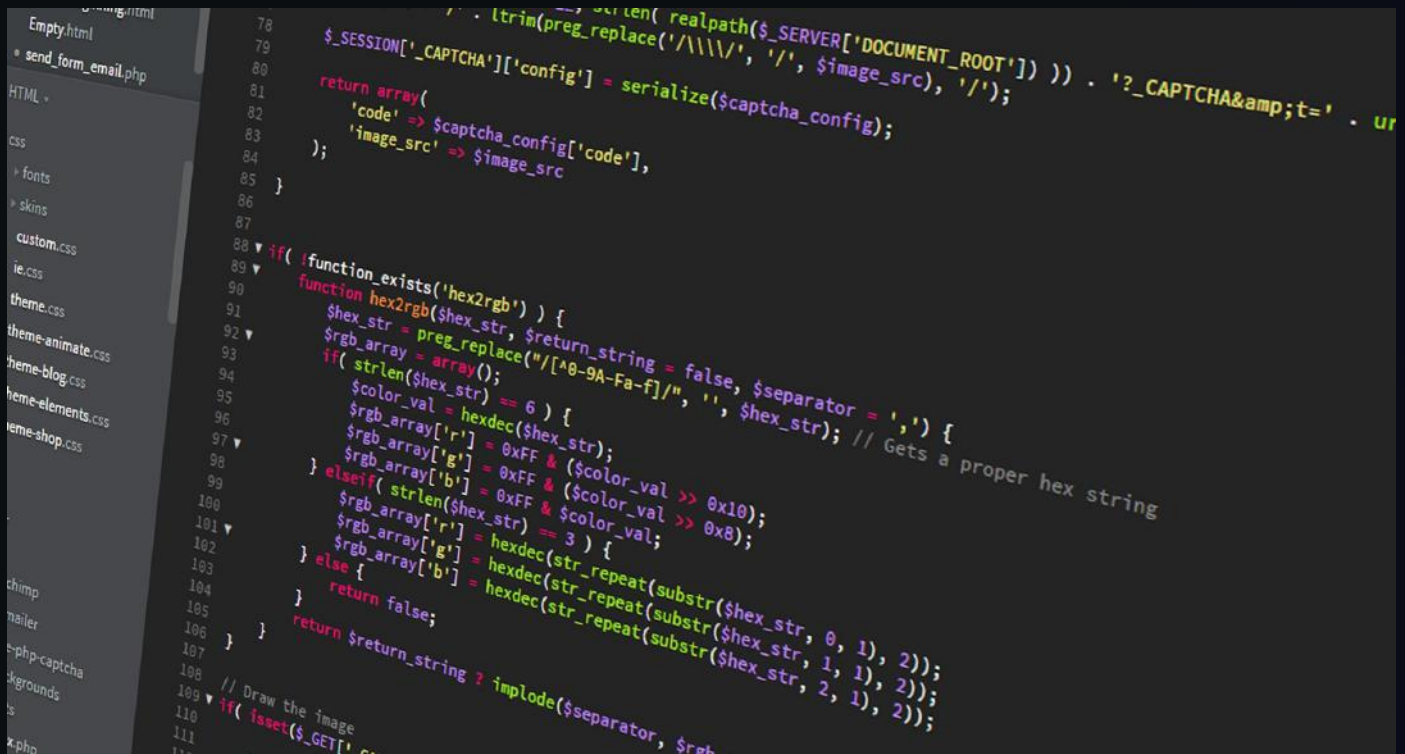
    function _nonReentrantAfter() private {
        // By storing the original value once again, a refund is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = _NOT_ENTERED;
    }
}

// File: ethPepeIco.sol

pragma solidity 0.8.20;
```

Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.



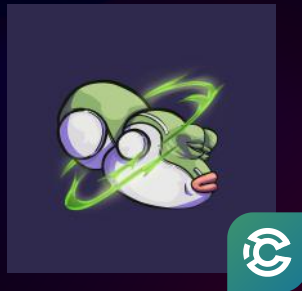
Type of check	Description
Mobile friendly?	● The website is mobile friendly
Contains jQuery errors?	● The website does not contain jQuery errors
Is SSL secured?	● The website is SSL secured
Contains spelling errors?	● The website does not contain spelling errors

Certificate of Proof

● Not KYC verified by Coinsult

Pepe ETH Ico

Audited by Coinsult.net



Date: 5 May 2025

✓ Advanced Manual Smart Contract Audit

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.


Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

End of report

Smart Contract Audit

 CoinsultAudits

 info@coinsult.net

 coinsult.net

Request your smart contract audit / KYC

t.me/coinsult_tg