FAIRYPROOF

Pepe Token

AUDIT REPORT

Version 1.0.0

Serial No. 2023082100012021

Presented by Fairyproof

August 21, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Pepe token issuance project.

Audit Start Time:

August 21, 2023

Audit End Time:

August 21, 2023

Audited Source File's Address:

https://etherscan.io/token/0x6982508145454ce325ddbe47a25d4ec3d2311933#code

The goal of this audit is to review Pepe's solidity implementation for its token issuance function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Pepe team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code Review, Including:

- Project Diagnosis

Understanding the size, scope and functionality of your project’s source code based on the specifications, sources, and instructions provided to Fairyproof.

- Manual Code Review

Reading your source code line-by-line to identify potential vulnerabilities.

- Specification Comparison

Determining whether your project’s code successfully and efficiently accomplishes or executes its functions according to the specifications, sources, and instructions provided to Fairyproof.

2. Testing and Automated Analysis, Including:

- Test Coverage Analysis

Determining whether the test cases cover your code and how much of your code is exercised or executed when test cases are run.

- Symbolic Execution

Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.

3. Best Practices Review

Reviewing the source code to improve maintainability, security, and control based on the latest established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following source(s) of truth about how the token issuance function should work:

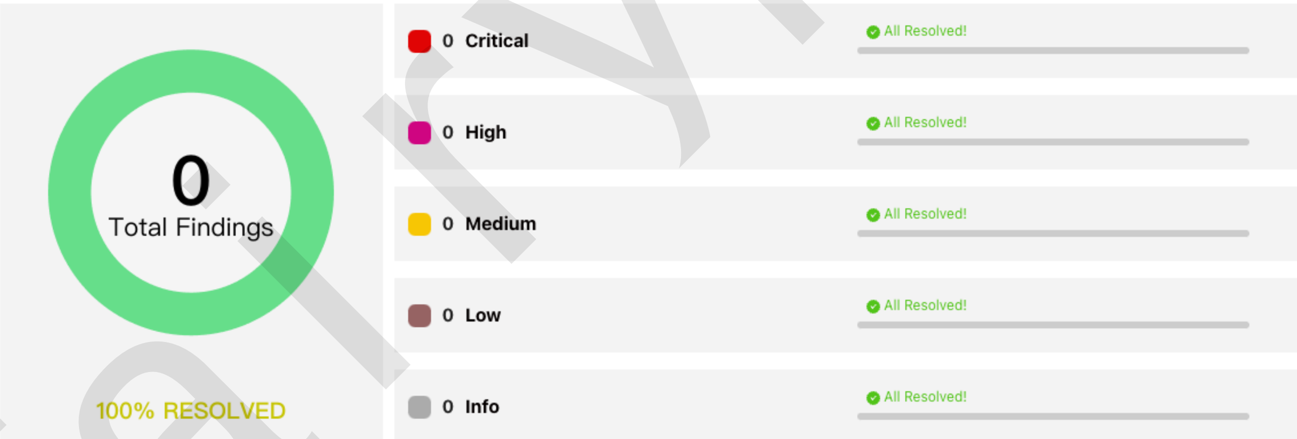
Website:https://www.pepe.vip/

Source Code: https://etherscan.io/token/0x6982508145454ce325ddbe47a25d4ec3d2311933#code

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Pepe team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023082100012021	Fairyproof Security Team	Aug 21, 2023 - Aug 21, 2023	Passed



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, no issues were uncovered.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to Pepe

PEPE is a deflationary memecoin launched on Ethereum. The cryptocurrency was created as a tribute to the Pepe the Frog internet meme, created by Matt Furie, which gained popularity in the early 2000s.
The project aims to capitalize on the popularity of meme coins, like Shiba Inu and Dogecoin, and strives to establish itself as one of the top meme-based cryptocurrencies. PEPE appeals to the cryptocurrency community by instituting a no-tax policy, a redistributive system rewarding long-term stakers, and a burning mechanism to maintain scarcity of the PEPE coin.
In late April to May 2023, the explosive surge of PEPE caused its market cap to reach a high of $1.6 billion at one point, minting millionaires out of early holders and attracting a strong community of like-minded followers. It has induced what some may dub a "memecoin season," causing other memecoins — some launched within hours — to go on spectacular pumps and just as astounding dumps. It remains to be seen if PEPE and other memecoins will go on to new highs, although that is certainly the hope of many bagholders.

The PEPE roadmap features three phases, where phase one includes listing on CoinMarketCap, and getting $PEPE trending on Twitter, while phase two includes listing on centralized exchanges (CEXs) and phase three includes “tier 1” exchange listings and what the team terms a “meme takeover.”

The above description is quoted from relevant documents of Pepe.

04. Major functions of audited code

The audited code mainly implements a token issuance function. Here are the details:

- Blockchain: Ethereum
- Token Standard: ERC-20
- Token Address: 0x6982508145454ce325ddbe47a25d4ec3d2311933
- Token Name: Pepe
- Token Symbol: PEPE
- Decimals: 18
- Current Supply: 420,689,899,999,995
- Max Supply: 420,690,000,000,000
- Burnable: Yes
- Blacklist: Yes

Note:

At this point, the owner of the contract is the zero address.

05. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Access Control
- Admin Rights
- Arithmetic Precision
- Code Improvement
- Contract Upgrade/Migration
- Delete Trap
- Design Vulnerability
- DoS Attack
- EOA Call Trap
- Fake Deposit
- Function Visibility
- Gas Consumption
- Implementation Vulnerability
- Inappropriate Callback Function
- Injection Attack
- Integer Overflow/Underflow
- IsContract Trap
- Miner's Advantage
- Misc
- Price Manipulation
- Proxy selector clashing
- Pseudo Random Number
- Re-entrancy Attack
- Replay Attack
- Rollback Attack
- Shadow Variable
- Slot Conflict
- Token Issuance
- Tx.origin Authentication
- Uninitialized Storage Pointer

06. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

07. Major areas that need attention

Based on the provided source code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Function Implementation

We checked whether or not the functions were correctly implemented.
We didn't find issues or risks in these functions or areas at the time of writing.

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by owner or administrator
We didn't find issues or risks in these functions or areas at the time of writing.

- Token Issuance & Transfer

We examined token issuance and transfers for situations that could harm the interests of holders.
We didn't find issues or risks in these functions or areas at the time of writing.

- State Update

We checked some key state variables which should only be set at initialization.
We didn't find issues or risks in these functions or areas at the time of writing.

- Asset Security

We checked whether or not all the functions that transfer assets were safely handled.
We didn't find issues or risks in these functions or areas at the time of writing.

- Miscellaneous

We checked the code for optimization and robustness.
We didn't find issues or risks in these functions or areas at the time of writing.

08. issues by severity

- N/A

09. Issue descriptions

- N/A

10. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- N/A

11. Appendices

11.1 Unit Test

1. PepeToken.t.js

```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("PepeToken Test", function () {
  let owner, addr1, addr2, uniswapV2PairMock;
  const totalSupply = ethers.parseEther("420690000000000")
  const AddressZero = "0x0000000000000000000000000000000000000000"


  async function deployToken() {
    [owner, addr1, addr2, uniswapV2PairMock] = await ethers.getSigners();
    const PepeToken = await ethers.getContractFactory("PepeToken");
    const instance = await PepeToken.deploy(totalSupply);
    return { instance };
  }

  describe("Deployment test", function () {
    it("Should set the correct metadata", async function () {
      const { instance } = await deployToken();

      // ERC20 metadata
      expect(await instance.totalSupply()).equal(totalSupply);
      expect(await instance.balanceOf(owner.address)).equal(totalSupply);
      expect(await instance.name()).equal("Pepe");
      expect(await instance.symbol()).equal("PEPE");
      expect(await instance.decimals()).equal(18);

      // PepeToken metadata
      expect(await instance.limited()).equal(false);
      expect(await instance.maxHoldingAmount()).equal(0);
      expect(await instance.minHoldingAmount()).equal(0);
      expect(await instance.uniswapV2Pair()).equal(AddressZero);
```

```
    });
  });

  describe("Function beforeTokenTransfer test", function () {
    it("Only owner can transfer until uniswapV2Pair is set", async function () {
      const { instance } = await deployToken();

      await expect(instance.connect(addr1).transfer(addr2.address,
1)).to.be.revertedWith("trading is not started");
      await instance.setRule(false, uniswapV2PairMock, 0, 0)
      await instance.transfer(addr1.address, 1);
      await instance.connect(addr1).transfer(addr2.address, 1);
    });

    it("Should respect maxHoldingAmount and minHoldingAmount", async () => {
      const { instance } = await deployToken();

      await instance.setRule(true, uniswapV2PairMock.address, 5000, 1000);

      await instance.transfer(uniswapV2PairMock.address, 6000)
      await expect(instance.connect(uniswapV2PairMock).transfer(addr1.address,
6000)).to.revertedWith("Forbid");
      await expect(instance.connect(uniswapV2PairMock).transfer(addr1.address,
500)).to.revertedWith("Forbid");
      await instance.connect(uniswapV2PairMock).transfer(addr1.address, 1200)
    });
  });

  describe("Transactions test", function () {
    it("Should transfer tokens between accounts", async function () {
      const { instance } = await deployToken();
      const transferAmount = 5000;

      await expect(instance.transfer(addr1.address, transferAmount))
        .be.emit(instance, "Transfer").withArgs(owner.address, addr1.address,
transferAmount);
      expect(await instance.balanceOf(addr1.address)).to.equal(transferAmount);
    });

    it("Should be failed if sender doesn’t have enough tokens", async function () {
      const { instance } = await deployToken();
      const initialOwnerBalance = await instance.balanceOf(owner.address);
      await expect(instance.connect(addr1).transfer(owner.address,
1)).to.revertedWith("ERC20: transfer amount exceeds balance");
      expect(await instance.balanceOf(owner.address)).to.equal(initialOwnerBalance);
    });

    it("Should be failed if sender transfer to zero address", async function () {
      const { instance } = await deployToken();
      const transferAmount = 5000;
      await expect(instance.transfer(AddressZero, transferAmount)).to.revertedWith("ERC20:
transfer to the zero address");
      await instance.approve(owner.address, transferAmount);
```

```
      await expect(instance.transferFrom(owner.address, AddressZero,
transferAmount)).to.revertedWith("ERC20: transfer to the zero address");
    });

    it("Should be successful if sender transfer to himself", async function () {
      const { instance } = await deployToken();
      const transferAmount = 5000;

      await expect(instance.transfer(owner.address, transferAmount))
        .be.emit(instance, "Transfer").withArgs(owner.address, owner.address,
transferAmount);
      await instance.approve(owner.address, transferAmount);
      await expect(instance.transferFrom(owner.address, owner.address, transferAmount))
        .be.emit(instance, "Transfer").withArgs(owner.address, owner.address,
transferAmount);
      expect(await instance.balanceOf(owner.address)).to.equal(totalSupply);
    });

    it("Should be successful if sender transfer zero amount", async function () {
      const { instance } = await deployToken();

      await expect(instance.transfer(addr1.address, 0))
        .be.emit(instance, "Transfer").withArgs(owner.address, addr1.address, 0);
      await expect(instance.transferFrom(owner.address, addr1.address, 0))
        .be.emit(instance, "Transfer").withArgs(owner.address, addr1.address, 0);
      expect(await instance.balanceOf(owner.address)).to.equal(totalSupply);
    });

    it("TransferFrom should need enough allowance", async function () {
      const { instance } = await deployToken();
      const transferAmount = 5000;

      await expect(instance.transferFrom(owner.address, addr1.address,
transferAmount)).to.revertedWith("ERC20: transfer amount exceeds allowance")
      await instance.approve(owner.address, transferAmount);
      await expect(instance.transferFrom(owner.address, addr1.address, transferAmount))
        .be.emit(instance, "Transfer").withArgs(owner.address, addr1.address,
transferAmount);
      expect(await instance.balanceOf(addr1.address)).to.equal(transferAmount);

      await instance.connect(addr1).approve(owner.address, transferAmount);
      await instance.transferFrom(addr1.address, owner.address, transferAmount)
      expect(await instance.balanceOf(addr1.address)).to.equal(0);
    });
  });


  describe("Burn test", function () {
    it("Allows users to burn their own tokens", async function () {
      const { instance } = await deployToken();

      await instance.setRule(true, uniswapV2PairMock.address, 5000, 1000);
      await instance.transfer(addr1.address, 1000);
```

```
      expect(await instance.balanceOf(addr1.address)).to.equal(1000);
      await instance.connect(addr1).burn(1000);
      expect(await instance.balanceOf(addr1.address)).to.equal(0);
      expect(await instance.totalSupply()).equal(totalSupply - (BigInt("1000")));
    });
  });

  describe("Allowance test", function () {
    it("Should update the allowance when approving", async function () {
      const { instance } = await deployToken();
      const approveAmount = 1000

      await expect(instance.approve(addr1.address, approveAmount))
        .to.emit(instance, "Approval").withArgs(owner.address, addr1.address,
approveAmount);
      const allowance = await instance.allowance(owner.address, addr1.address);
      expect(allowance).to.equal(approveAmount);
      // increse allowance again
      await expect(instance.increaseAllowance(addr1.address, approveAmount))
        .to.emit(instance, "Approval").withArgs(owner.address, addr1.address, approveAmount
* 2);
      expect(await instance.allowance(owner.address, addr1.address)).to.equal(approveAmount
* 2);

      // decrease allowance
      await expect(instance.decreaseAllowance(addr1.address, approveAmount))
        .to.emit(instance, "Approval").withArgs(owner.address, addr1.address,
approveAmount);
    });
  });

  describe("Ownership test", function () {
    it("Should transfer and renounce ownership correctly", async function () {
      const { instance } = await deployToken();

      expect(await instance.owner()).to.equal(owner.address);
      await instance.transferOwnership(addr1.address);
      expect(await instance.owner()).to.equal(addr1.address);

      await instance.connect(addr1).renounceOwnership();
      expect(await instance.owner()).to.equal(AddressZero);
    });

    it("Should lose ownership if the owner renounces ownership", async function () {
      const { instance } = await deployToken();

      expect(await instance.owner()).to.equal(owner.address);
      await instance.renounceOwnership();
      expect(await instance.owner()).to.equal(AddressZero);

      // lose ownership
      await expect(instance.blacklist(addr1.address, true)).to.revertedWith("Ownable:
caller is not the owner");
```

```
      await expect(instance.setRule(true, AddressZero, AddressZero,
AddressZero)).to.revertedWith("Ownable: caller is not the owner");
    });
  });

  describe("BlackLists test", function () {
    it("Users in blacklist should not be able to transfer", async function () {
      const { instance } = await deployToken();
      const transferAmount = 5000;

      await instance.blacklist(addr1.address, true);
      await expect(instance.transfer(addr1.address,
transferAmount)).to.revertedWith("Blacklisted");
      await expect(instance.connect(addr1).transfer(owner.address,
transferAmount)).to.revertedWith("Blacklisted");
      expect(await instance.balanceOf(addr1.address)).to.equal(0);

      await instance.blacklist(addr1.address, false);
      await instance.transfer(addr1.address, transferAmount)
      expect(await instance.balanceOf(addr1.address)).to.equal(transferAmount);
    });

    it("The owner can manage the blacklist himself", async function () {
      const { instance } = await deployToken();
      const transferAmount = 5000;

      await instance.blacklist(owner.address, true);
      await expect(instance.transfer(addr1.address,
transferAmount)).to.revertedWith("Blacklisted");
      await instance.blacklist(owner.address, false);
      await instance.transfer(addr1.address, transferAmount)
    });
  });
});
```

2. output:

```
  PepeToken Test
    Deployment test
      ✓ Should set the correct metadata (919ms)
    Function beforeTokenTransfer test
      ✓ Only owner can transfer until uniswapV2Pair is set (82ms)
      ✓ Should respect maxHoldingAmount and minHoldingAmount (52ms)
    Transactions test
      ✓ Should transfer tokens between accounts
      ✓ Should be failed if sender doesn’t have enough tokens
      ✓ Should be failed if sender transfer to zero address
      ✓ Should be successful if sender transfer to himself
      ✓ Should be successful if sender transfer zero amount
```

```
    ✓ TransferFrom should need enough allowance (52ms)
  Burn test
    ✓ Allows users to burn their own tokens
  Allowance test
    ✓ Should update the allowance when approving
  Ownership test
    ✓ Should transfer and renounce ownership correctly
    ✓ Should lose ownership if the owner renounces ownership
  BlackLists test
    ✓ Users in blacklist should not be able to transfer (40ms)
    ✓ The owner can manage the blacklist himself


15 passing (1s)
```

11.2 External Functions Check Points

1. File: contracts/PepeToken.sol

(Empty fields in the table represent things that are not required or relevant)

contract: PepeToken is Ownable, ERC20

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	blacklist(address,bool)	external		onlyOwner		Passed	
2	setRule(bool,address,uint256,uint256)	external		onlyOwner		Passed	
3	burn(uint256)	external			Yes	Passed	
4	name()	public	view			Passed	
5	symbol()	public	view			Passed	
6	decimals()	public	view			Passed	
7	totalSupply()	public	view			Passed	
8	balanceOf(address)	public	view			Passed	
9	transfer(address,uint256)	public			Yes	Passed	
10	allowance(address,address)	public	view			Passed	
11	approve(address,uint256)	public			Yes	Passed	
12	transferFrom(address,address,uint256)	public			Yes	Passed	
13	increaseAllowance(address,uint256)	public			Yes	Passed	
14	decreaseAllowance(address,uint256)	public			Yes	Passed	
15	owner()	public	view			Passed	
16	renounceOwnership()	public		onlyOwner		Passed	
17	transferOwnership(address)	public		onlyOwner		Passed	

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech