OCD Token

AUDIT REPORT

Version 1.0.0

Serial No. 20231014000012016

Presented by Fairyproof

October 14, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the On-Chain Dynamics token issuance project.

Audit Start Time:

October 13, 2023

Audit End Time:

October 14, 2023

Audited Source File's Address:

https://etherscan.io/address/0x017e9db34fc69af0dc7c7b4b33511226971cddc7#code

The goal of this audit is to review On-Chain Dynamics Token'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 On-Chain Dynamics Token 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.

Fairyproof

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.

Fairyproof

— 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.onchaindynamics.io/

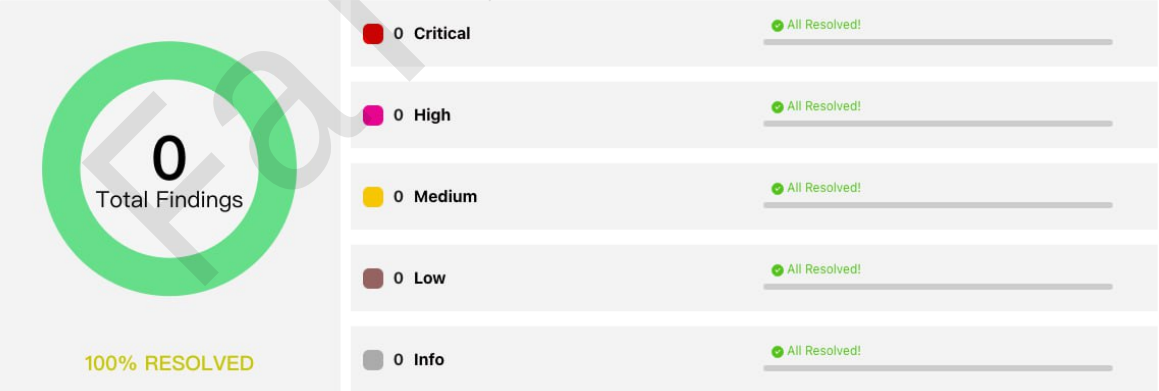
Whitepaper:https://www.onchaindynamics.io/OCD_LitepaperV2.pdf

Source Code:
https://etherscan.io/address/0x017e9db34fc69af0dc7c7b4b33511226971cddc7#code

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the On-Chain Dynamics Token team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023101400012016	Fairyproof Security Team	Oct 13, 2023 - Oct 14, 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.

Fairyproof

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 On-Chain Dynamics Token

Onchain Dynamics, encapsulating the intricate interactions among nodes, consensus mechanisms, smart contracts, decentralized applications (DApps), scalability solutions, and governance processes, stand as the vital life force of blockchain technology.

The above description is quoted from relevant documents of On-Chain Dynamics Token.

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: 0x017e9db34fc69af0dc7c7b4b33511226971cddc7
- Token Name: On-Chain Dynamics
- Token Symbol: OCD
- Decimals: 18
- Current Supply: 1,000,000,000
- Max Supply: 1,000,000,000
- Taxable: Yes

Note:

Transaction fees will be charged when traders trade specific tokens or add liquidity for the specific token pairs. The charged transaction fees will be converted to the specific token pairs and added to the liquidity. The admin can change the transaction fee's rate.

Fairyproof

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

Fairyproof

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.

Fairyproof

- 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.

- Consider managing the owner's access control with great care and transfering it to a multi-sig wallet or DAO when necessary.

Fairyproof

11. Appendices

11.1 Unit Test

1. OCD.t.js

```
const {
  loadFixture,
  impersonateAccount
} = require("@nomicfoundation/hardhat-toolbox/network-helpers");
const { expect } = require("chai");
const { ZeroAddress } = require("ethers");
const { ethers } = require("hardhat");

describe("OCD Token Test", function () {
  let addr1, addr2;
  const totalSupply = ethers.parseEther("1000000000")
  const Depoloyer = "0xeCAB3064B0FCa52fdcc8422280a927EF8f51fE8D"
  const DexRouter = "0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D"
  const OCD = "0x017E9Db34fC69Af0dC7c7b4b33511226971cDdc7"
  const WETH = "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2"
  const MarketWallet = "0xdBBa71D308125218B1cD0fa4f93662EbDc28b43D"
  const minSwapAmount = totalSupply / BigInt(4000)
  const percentDivider = BigInt(100)
  const marketFeeOnBuy = BigInt(1)
  const marketFeeOnSell = BigInt(9)
  const addr1Balance = ethers.parseEther("100")

  const getTotalBuyFeePerTx = (amount) => (amount * marketFeeOnBuy) /
percentDivider
  const getTotalSellFeePerTx = (amount) => (amount * marketFeeOnSell) /
percentDivider

  async function deployToken() {
    [addr1, addr2] = await ethers.getSigners();
    // const OCD = await ethers.getContractFactory("OCD");
    // const instance = await OCD.deploy();
    const instance = await ethers.getContractAt("OCD", OCD);
    await impersonateAccount(Depoloyer);
    const DepoloySigner = await ethers.getSigner(Depoloyer)
    await instance.connect(DepoloySigner).transfer(addr1.address, addr1Balance);

    const transferOwnership = async () => {
      await impersonateAccount(Depoloyer);
      const DepoloySigner = await ethers.getSigner(Depoloyer)
      await instance.connect(DepoloySigner).transferOwnership(addr1.address);
      expect(await instance.owner()).to.equal(addr1.address);
    }
```

Fairyproof

```
    return { instance, DepoloySigner, transferOwnership };
  }

  describe("Read contract test", function () {
    it("Should have the correct erc20 metadata", async function () {
      const { instance } = await loadFixture(deployToken);

      expect(await instance.name()).to.equal("On-Chain Dynamics");
      expect(await instance.symbol()).to.equal("OCD");
      expect(await instance.decimals()).to.equal(18);
      expect(await instance.owner()).to.equal(Depoloyer);
      expect(await instance.totalSupply()).to.equal(totalSupply);
      expect(await instance.balanceOf(Depoloyer)).to.lt(totalSupply);
    });

    it("Should have the correct state", async function () {
      const { instance } = await loadFixture(deployToken);

      expect(await instance.dexRouter()).to.equal(DexRouter);
      expect(await instance.distributeAndLiquifyStatus()).to.equal(true);
      expect(await instance.feesStatus()).to.equal(true);
      expect(await instance.percentDivider()).to.equal(percentDivider);
      expect(await instance.marketWallet()).to.equal(MarketWallet);
      expect(await instance.minSwapAmount()).to.equal(minSwapAmount);
      expect(await instance.marketFeeOnBuy()).to.equal(marketFeeOnBuy);
      expect(await instance.marketFeeOnSell()).to.equal(marketFeeOnSell);

      expect(await instance.isExcludedFromFee(Depoloyer)).to.equal(true);
      expect(await instance.isExcludedFromFee(OCD)).to.equal(true);
    });

    it("Should have the correct fee getter", async function () {
      const { instance } = await loadFixture(deployToken);

      const amount = BigInt(100)
      expect(await
instance.totalBuyFeePerTx(amount)).to.equal(getTotalBuyFeePerTx(amount));
      expect(await
instance.totalSellFeePerTx(amount)).to.equal(getTotalSellFeePerTx(amount));
    });
  });

  describe("Transactions between eoa accounts test", function () {
    it("Should transfer tokens between accounts", async function () {
      const { instance } = await loadFixture(deployToken);
      expect(await instance.balanceOf(addr1.address)).to.equal(addr1Balance);
      const transferAmount = BigInt(5000);
      await expect(instance.transfer(addr2.address, transferAmount))
        .be.emit(instance, "Transfer").withArgs(addr1.address, addr2.address,
transferAmount);
      expect(await instance.balanceOf(addr2.address)).to.equal(transferAmount);
      expect(await instance.balanceOf(addr1.address)).to.equal(addr1Balance -
transferAmount);
    });
```

Fairyproof

```
    it("Should be failed if sender doesn't have enough tokens", async function ()
{
      const { instance } = await loadFixture(deployToken);
      expect(await instance.balanceOf(addr2.address)).to.equal(0);
      await expect(instance.connect(addr2).transfer(addr1.address,
1)).to.reverted;
      expect(await instance.balanceOf(addr1.address)).to.equal(addr1Balance);
    });

    it("Should be failed if sender transfer to or transfer from zero address",
async function () {
      const { instance } = await loadFixture(deployToken);
      const transferAmount = BigInt(5000);
      await expect(instance.transfer(ZeroAddress,
transferAmount)).to.revertedWith("OCD: transfer to the zero address");
      await expect(instance.transferFrom(addr1.address, ZeroAddress,
transferAmount)).to.revertedWith("OCD: transfer to the zero address");
      await expect(instance.transferFrom(ZeroAddress, addr1.address,
transferAmount)).to.revertedWith("OCD: transfer from the zero address");
      await expect(instance.transferFrom(addr1.address, addr2.address,
0)).to.revertedWith("OCD: Amount must be greater than zero");
    });

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

      await expect(instance.transfer(addr2.address, transferAmount))
        .be.emit(instance, "Transfer").withArgs(addr1.address, addr2.address,
transferAmount);
      await instance.approve(addr1.address, transferAmount);
      await expect(instance.transferFrom(addr1.address, addr1.address,
transferAmount))
        .be.emit(instance, "Transfer").withArgs(addr1.address, addr1.address,
transferAmount);
      expect(await instance.balanceOf(addr1.address)).to.equal(addr1Balance -
transferAmount);
    });

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

      const transferAmount = BigInt(5000);

      await instance.transfer(addr2.address, transferAmount)
      await expect(instance.transferFrom(addr2.address, addr1.address,
transferAmount)).to.reverted;

      await instance.connect(addr2).approve(addr1.address, transferAmount);
      await expect(instance.transferFrom(addr2.address, addr1.address,
transferAmount))
        .be.emit(instance, "Transfer").withArgs(addr2.address, addr1.address,
transferAmount);
      expect(await instance.balanceOf(addr1.address)).to.equal(addr1Balance);
    });
```

Fairyproof

```
  });

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

      await expect(instance.approve(addr2.address, approveAmount))
        .to.emit(instance, "Approval").withArgs(addr1.address, addr2.address,
approveAmount);
      const allowance = await instance.allowance(addr1.address, addr2.address);
      expect(allowance).to.equal(approveAmount);
      // increse allowance again
      await expect(instance.increaseAllowance(addr2.address, approveAmount))
        .to.emit(instance, "Approval").withArgs(addr1.address, addr2.address,
approveAmount * BigInt(2));
      expect(await instance.allowance(addr1.address,
addr2.address)).to.equal(approveAmount * BigInt(2));
      // decrease allowance
      await expect(instance.decreaseAllowance(addr2.address, approveAmount))
        .to.emit(instance, "Approval").withArgs(addr1.address, addr2.address,
approveAmount);
    });

    it("Should underflow when decreasing allowance below zero", async function ()
{
      const { instance } = await loadFixture(deployToken);
      const approveAmount = ethers.parseEther("1000");
      await instance.approve(addr2.address, approveAmount);

      await expect(instance.decreaseAllowance(addr2.address, approveAmount + 1n))
        .to.reverted;
      expect(await instance.allowance(addr1.address,
addr2.address)).to.equal(approveAmount);
    });
  });

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

      expect(await instance.owner()).to.equal(Depoloyer);
      await impersonateAccount(Depoloyer);
      const DepoloySigner = await ethers.getSigner(Depoloyer)
      await instance.connect(DepoloySigner).transferOwnership(addr1.address);
      expect(await instance.owner()).to.equal(addr1.address);

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

  describe("Ownable functions test", function () {
    it("Only owner can call function setIncludeOrExcludeFromFee", async function
() {
      const { instance, transferOwnership } = await loadFixture(deployToken);
```

Fairyproof

```
      await expect(instance.setIncludeOrExcludeFromFee(addr1.address,
true)).be.revertedWith("Ownable: caller is not the owner")
      await transferOwnership()
      expect(await instance.isExcludedFromFee(addr1.address)).to.equal(false);
      await expect(instance.setIncludeOrExcludeFromFee(addr1.address, true))
        .be.emit(instance, "ExcludeFromFee")
        .withArgs(addr1.address, true);
      expect(await instance.isExcludedFromFee(addr1.address)).to.equal(true);
    });

    it("Only owner can call function updateSwapAmount", async function () {
      const { instance, transferOwnership } = await loadFixture(deployToken);
      await expect(instance.updateSwapAmount(1000)).be.revertedWith("Ownable:
caller is not the owner")
      await transferOwnership()
      expect(await instance.minSwapAmount()).to.equal(minSwapAmount);
      await expect(instance.updateSwapAmount(1000))
        .be.emit(instance, "NewSwapAmount").withArgs(ethers.parseEther("1000"));
      expect(await instance.minSwapAmount()).to.equal(ethers.parseEther("1000"));
    });

    it("Only owner can call function updateBuyFee", async function () {
      const { instance, transferOwnership } = await loadFixture(deployToken);
      await expect(instance.updateBuyFee(2)).be.revertedWith("Ownable: caller is
not the owner")
      await transferOwnership()
      expect(await instance.marketFeeOnBuy()).to.equal(marketFeeOnBuy);
      await expect(instance.updateBuyFee(2))
        .be.emit(instance, "FeeUpdated").withArgs(2);
      expect(await instance.marketFeeOnBuy()).to.equal(2);
    });

    it("Only owner can call function updateSellFee", async function () {
      const { instance, transferOwnership } = await loadFixture(deployToken);
      await expect(instance.updateSellFee(10)).be.revertedWith("Ownable: caller
is not the owner")
      await transferOwnership()
      expect(await instance.marketFeeOnSell()).to.equal(marketFeeOnSell);
      await expect(instance.updateSellFee(10))
        .be.emit(instance, "FeeUpdated").withArgs(10);
      expect(await instance.marketFeeOnSell()).to.equal(10);
    });

    it("Only owner can call function setDistributionStatus", async function () {
      const { instance, transferOwnership } = await loadFixture(deployToken);
      await
expect(instance.setDistributionStatus(false)).be.revertedWith("Ownable: caller is
not the owner")
      await transferOwnership()
      await expect(instance.setDistributionStatus(true)).be.revertedWith("Value
must be different from current state")
      expect(await instance.distributeAndLiquifyStatus()).to.equal(true);
      await expect(instance.setDistributionStatus(false))
        .be.emit(instance, "DistributionStatus").withArgs(false);
      expect(await instance.distributeAndLiquifyStatus()).to.equal(false);
    });
```

Fairyproof

```
    it("Only owner can call function enableOrDisableFees", async function () {
      const { instance, transferOwnership } = await loadFixture(deployToken);
      await expect(instance.enableOrDisableFees(false)).be.revertedWith("Ownable:
caller is not the owner")
      await transferOwnership()
      await expect(instance.enableOrDisableFees(true)).be.revertedWith("Value
must be different from current state")
      expect(await instance.feesStatus()).to.equal(true);
      await expect(instance.enableOrDisableFees(false))
        .be.emit(instance, "FeeStatus").withArgs(false);
      expect(await instance.feesStatus()).to.equal(false);
    });

    it("Only owner can call function updatemarketWallet", async function () {
      const { instance, transferOwnership } = await loadFixture(deployToken);
      await
expect(instance.updatemarketWallet(addr2.address)).be.revertedWith("Ownable:
caller is not the owner")
      await transferOwnership()
      await
expect(instance.updatemarketWallet(ZeroAddress)).be.revertedWith("Ownable: new
marketWallet is the zero address")
      expect(await instance.marketWallet()).to.equal(MarketWallet);
      await expect(instance.updatemarketWallet(addr2.address))
        .be.emit(instance, "marketWalletUpdated").withArgs(addr2.address,
MarketWallet);
      expect(await instance.marketWallet()).to.equal(addr2.address);
    });
  });

  describe("Send ETH and withdrawETH test", function () {
    it("Contract can receive ETH", async function () {
      const initialBalance = await ethers.provider.getBalance(OCD)
      const amount = ethers.parseEther("1.0");
      await addr1.sendTransaction({
        to: OCD,
        value: amount
      });
      expect(await ethers.provider.getBalance(OCD)).to.gt(initialBalance);
    });

    it("Only owner can call withdrawETH", async function () {
      const { instance, transferOwnership } = await loadFixture(deployToken);
      await expect(instance.withdrawETH(100)).be.revertedWith("Ownable: caller is
not the owner")
      await transferOwnership()
      const initialBalance = await ethers.provider.getBalance(OCD)
      await expect(instance.withdrawETH(initialBalance +
BigInt(100))).be.revertedWith("Invalid Amount")
      const amount = ethers.parseEther("1.0");
      await addr1.sendTransaction({
        to: OCD,
        value: amount
      });
      await expect(instance.withdrawETH(100))
```

Fairyproof

```
        .be.emit(instance, "Transfer").withArgs(OCD, addr1.address, 100);
    });
  });

  describe("Transfer fee unit test", function () {
    it("Add liquidity will take fee", async function () {
      const { instance } = await loadFixture(deployToken);

      const dexPairAddress = await instance.dexPair()
      expect(await instance.isExcludedFromFee(dexPairAddress)).to.equal(false)
      const initialDexPairBalance = await instance.balanceOf(dexPairAddress)
      const initialOcdBalance = await instance.balanceOf(OCD)

      const DexRouterInstance = await ethers.getContractAt("UniswapV2Router02",
DexRouter);
      const amountETH = ethers.parseEther("1");
      await instance.approve(DexRouter, addr1Balance);
      await DexRouterInstance.addLiquidityETH(
        OCD,
        addr1Balance,
        0,
        0,
        addr1.address,
        Date.now() + 1000 * 60 * 10,
        { value: amountETH }
      )

      const allFee = getTotalSellFeePerTx(addr1Balance)
      expect(await instance.balanceOf(addr1.address)).to.equal(0);
      expect(await instance.balanceOf(OCD)).to.equal(initialOcdBalance + allFee);
      expect(await
instance.balanceOf(dexPairAddress)).to.equal(initialDexPairBalance + addr1Balance
- allFee);
    });

    it("Tansfer to dexPair will take fee", async function () {
      const { instance } = await loadFixture(deployToken);

      const dexPairAddress = await instance.dexPair()
      expect(await instance.isExcludedFromFee(dexPairAddress)).to.equal(false)
      const initialDexPairBalance = await instance.balanceOf(dexPairAddress)
      const initialOcdBalance = await instance.balanceOf(OCD)

      await instance.transfer(dexPairAddress, addr1Balance)
      const allFee = getTotalSellFeePerTx(addr1Balance)
      expect(await instance.balanceOf(addr1.address)).to.equal(0);
      expect(await instance.balanceOf(OCD)).to.equal(initialOcdBalance + allFee);
      expect(await
instance.balanceOf(dexPairAddress)).to.equal(initialDexPairBalance + addr1Balance
- allFee);
    });

    it("Remove liquidity will not take fee if router is excludedFromFee", async
function () {
      const { instance } = await loadFixture(deployToken);
```

Fairyproof

```
      const dexPairAddress = await instance.dexPair()
      expect(await instance.isExcludedFromFee(dexPairAddress)).to.equal(false)
      expect(await instance.balanceOf(addr2.address)).to.equal(0);

      const DexRouterInstance = await ethers.getContractAt("UniswapV2Router02",
DexRouter);
      const amountETH = ethers.parseEther("1");
      await instance.approve(DexRouter, addr1Balance);
      // add liquidiy first
      await DexRouterInstance.addLiquidityETH(OCD, addr1Balance, 0, 0,
addr1.address, Date.now() + 1000 * 60 * 10, { value: amountETH })
      const initialPairBalance = await instance.balanceOf(dexPairAddress)
      const initialOcdBalance = await instance.balanceOf(OCD)
      const dexPairInstance = await ethers.getContractAt("UniswapV2Pair",
dexPairAddress);
      const liquidityAmount = await dexPairInstance.balanceOf(addr1.address)
      expect(liquidityAmount).to.gt(0)

      // remove liqudity
      await dexPairInstance.approve(DexRouter, liquidityAmount)
      await DexRouterInstance.removeLiquidityETH(OCD, liquidityAmount, 0, 0,
addr2.address, Date.now() + 1000 * 60 * 10)
      const [reserve0, reserve1] = await dexPairInstance.getReserves()
      const totalLiquidity = await dexPairInstance.totalSupply()
      const amount0 = liquidityAmount * reserve0 / totalLiquidity
      const amount1 = liquidityAmount * reserve1 / totalLiquidity
      const allFee = getTotalBuyFeePerTx(amount0)
      expect(await instance.balanceOf(addr2.address)).to.equal(amount0);
      expect(await
instance.balanceOf(dexPairAddress)).to.equal(initialPairBalance - amount0);
    });

    it("Transfer include buy fee test", async () => {
      const { instance } = await loadFixture(deployToken);
      const dexPairAddress = await instance.dexPair()
      const DexRouterInstance = await ethers.getContractAt("UniswapV2Router02",
DexRouter);
      const dexPairInstance = await ethers.getContractAt("UniswapV2Pair",
dexPairAddress);
      const amountETH = ethers.parseEther("1");
      await instance.approve(DexRouter, addr1Balance);
      await DexRouterInstance.addLiquidityETH(
        OCD,
        addr1Balance,
        0,
        0,
        addr1.address,
        Date.now() + 1000 * 60 * 10,
        { value: amountETH }
      )

      const [reserve0, reserve1, blockTimestampLast] = await
dexPairInstance.getReserves()
      const amountIn = ethers.parseEther('0.5')
      const amountOut = await DexRouterInstance.getAmountOut(amountIn, reserve1,
reserve0)
```

Fairyproof

```
      const allFee = getTotalBuyFeePerTx(amountOut)
      const initialOcdBalance = await instance.balanceOf(OCD)
      await DexRouterInstance.swapExactETHForTokensSupportingFeeOnTransferTokens(
        0,
        [WETH, OCD],
        addr2.address,
        Date.now() + 1000 * 60 * 10,
        { value: amountIn }
      )
      expect(await instance.balanceOf(OCD)).to.equal(initialOcdBalance + allFee);
      expect(await instance.balanceOf(addr2.address)).to.be.equal(amountOut -
(allFee))
    })

    it("Function distributeAndLiquify will excute while contractTokenBalance
reach minSwapAmount", async () => {
      const { instance, DepoloySigner } = await loadFixture(deployToken);

      const DexRouterInstance = await ethers.getContractAt("UniswapV2Router02",
DexRouter);
      await instance.connect(DepoloySigner).setIncludeOrExcludeFromFee(Depoloyer,
false)
      const amountETH = ethers.parseEther("15");
      await instance.connect(DepoloySigner).approve(DexRouter, totalSupply /
BigInt(2));
      await DexRouterInstance.connect(DepoloySigner).addLiquidityETH(
        OCD,
        totalSupply / BigInt(2),
        0,
        0,
        addr1.address,
        Date.now() + 1000 * 60 * 10,
        { value: amountETH }
      )
      const contractTokenBalance = await instance.balanceOf(OCD)
      expect(contractTokenBalance).to.gt(minSwapAmount);

      // distributeAndLiquify will excute
      await instance.transfer(addr2.address, BigInt(100))
      expect(await instance.balanceOf(OCD)).to.equal(contractTokenBalance -
minSwapAmount);
    });

  });
});
```

Fairyproof

2. output:

```
 OCD Token Test
    Read contract test
      ✓ Should have the correct erc20 metadata (16711ms)
      ✓ Should have the correct state (2582ms)
      ✓ Should have the correct fee getter
    Transactions between eoa accounts test
      ✓ Should transfer tokens between accounts (1137ms)
      ✓ Should be failed if sender doesn't have enough tokens (58ms)
      ✓ Should be failed if sender transfer to or transfer from zero address
      ✓ Should be successful if sender transfer to himself, and will loose fees
(394ms)
      ✓ TransferFrom should need enough allowance (402ms)
    Allowance test
      ✓ Should update the allowance after approving (421ms)
      ✓ Should underflow when decreasing allowance below zero (44ms)
    Ownership test
      ✓ Should transfer and renounce ownership correctly
    Ownable functions test
      ✓ Only owner can call function setIncludeOrExcludeFromFee
      ✓ Only owner can call function updateSwapAmount
      ✓ Only owner can call function updateBuyFee
      ✓ Only owner can call function updateSellFee
      ✓ Only owner can call function setDistributionStatus
      ✓ Only owner can call function enableOrDisableFees
      ✓ Only owner can call function updatemarketWallet
    Send ETH and withdrawETH test
      ✓ Contract can receive ETH
      ✓ Only owner can call withdrawETH

      ✓ Add liquidity will take fee (12696ms)
      ✓ Tansfer to dexPair will take fee (43ms)
      ✓ Remove liquidity will not take fee if router is excludedFromFee (1681ms)
      ✓ Transfer include buy fee test (93ms)
      ✓ Function distributeAndLiquify will excute while contractTokenBalance
reach minSwapAmount (1625ms)


  25 passing (38s)
```

Fairyproof

11.2 External Functions Check Points

1. File: contracts/OCD.sol

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

contract: OCD is Context, IERC20, Ownable, ReentrancyGuard

Index	Function	Visibility	StateMutability	Permission Check	IsUserInterface	Unit Test	Notes
1	setIncludeOrExcludeFromFee(address,bool)	external		onlyOwner		Passed	
2	updateSwapAmount(uint256)	external		onlyOwner		Passed	
3	updateBuyFee(uint256)	external		onlyOwner		Passed	
4	updateSellFee(uint256)	external		onlyOwner		Passed	
5	setDistributionStatus(bool)	external		onlyOwner		Passed	
6	enableOrDisableFees(bool)	external		onlyOwner		Passed	
7	updatemarketWallet(address)	external		onlyOwner		Passed	
8	receive()	external	payable			Passed	
9	name()	public	view			Passed	
10	symbol()	public	view			Passed	
11	decimals()	public	view			Passed	
12	totalSupply()	public	view			Passed	
13	balanceOf(address)	public	view			Passed	
14	transfer(address,uint256)	public			Yes	Passed	
15	allowance(address,address)	public	view			Passed	
16	approve(address,uint256)	public			Yes	Passed	
17	transferFrom(address,address,uint256)	public			Yes	Passed	
18	increaseAllowance(address,uint256)	public			Yes	Passed	
19	decreaseAllowance(address,uint256)	public			Yes	Passed	
20	totalBuyFeePerTx(uint256)	public	view			Passed	
21	totalSellFeePerTx(uint256)	public	view			Passed	
22	withdrawETH(uint256)	external		onlyOwner		Passed	
23	owner()	public	view			Passed	
24	renounceOwnership()	public		onlyOwner		Passed	
25	transferOwnership(address)	public		onlyOwner		Passed	

Fairyproof

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