FAIRYPROOF

Zebra Dex

AUDIT REPORT

Version 1.0.0

Serial No. 2023102000012013

Presented by Fairyproof

October 20, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Zebra Dex project.

Audit Start Time:

October 19, 2023

Audit End Time:

October 20, 2023

Audited Source File's Address:

https://scrollscan.com/address/0xa63eb44c67813cad20A9aE654641ddc918412941#code

https://scrollscan.com/address/0x0122960d6e391478bfE8fB2408Ba412D5600f621#code

Audited Code's Github Repository:

https://github.com/zebra-xyz/contracts

Audited Code's Github Commit Number When Audit Started:

4e4f03b9ae4a10cc2e500a3fa33748607554ad57

Audited Code's Github Commit Number When Audit Ended:

6267fe296494536c31f751e375b644963a0ac783

Audited Source Files:

The source files audited include all the files as follows:

```
contracts/*.sol
```

The goal of this audit is to review Zebra's solidity implementation for its Dex 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 Zebra 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://zebra.xyz/

Whitepaper:https://zebra.gitbook.io/zebra-docs/

Source Code:

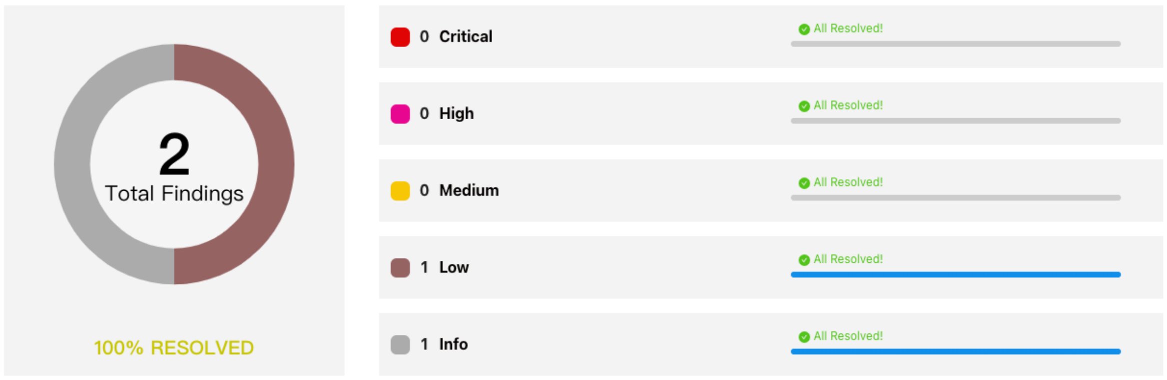
https://scrollscan.com/address/0xa63eb44c67813cad20A9aE654641ddc918412941#code

https://scrollscan.com/address/0x0122960d6e391478bfE8fB2408Ba412D5600f621#code

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023102000012013	Fairyproof Security Team	Oct 19, 2023 - Oct 20, 2023	Passed



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, one issue of low-severity and one issue of info-severity were uncovered. The Zebra team fixed all the issues.

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 Zebra

Zebra is a fully permissionless and composable decentralized exchange built on Scroll. It is committed to providing users with a one-stop liquidity service that is cheaper, easier to use, and more secure.

The above description is quoted from relevant documents of Zebra.

04. Major functions of audited code

The main function of the audit code is a UniswapV2-like decentralized exchange.
The transaction fee is three thousandths, and the platform’s share of the fee can be adjusted.

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 found one issue, for more details please refer to [FP-1] in "09. Issue description".

- 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 found one issue, for more details please refer to [FP-2] in "09. Issue description".

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Lack of Max Value for Parameter	Design Vulnerability	Low	✓ Fixed
FP-2	Confusing Parameter Name	Implementation Vulnerability	Info	✓ Fixed

09. Issue descriptions

[FP-1] Lack of Max Value for Parameter

Design Vulnerability Low ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

In `ZebraFactory.sol`, the `setFeeToRate` function didn't check the max value of `_rate`. In extreme cases, `rate` could be assigned with an extremely large value resulting in overflow issues in the `mintFee` function in `ZebraPair.sol`.

Recommendation:

Consider setting a max value for `_rate`.

Update:

The Zebra team set a max value of 100 for `_rate`.

Status:

The zebra team has fixed the issue.

[FP-2] Confusing Parameter Name

Implementation Vulnerability Info ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In the `price` function defined in `ZebraPair.sol`, the `baseDecimal` parameter was confusing. In general `decimal` or `decimals` means a token's decimal. For example 18 stands for 10 ** 18.

Recommendation:

Consider using `baseUnit` instead of `baseDecimal`

Update:

The Zebra team changed the parameter name.

Status:

The Zebra team has fixed the issue.

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.

11. Appendices

11.1 Unit Test

1. MockErc20.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MockERC20 is ERC20 {
    address private owner = msg.sender;
    constructor(string memory name, string memory symbol, uint256 initialSupply)
ERC20(name, symbol) public {
        _mint(msg.sender, initialSupply);
    }

    function mint(address to ,uint amount) external {
        require(msg.sender == owner,"not owner");
        _mint(to,amount);
    }

    function burn(address to ,uint amount) external {
        require(msg.sender == owner,"not owner");
        _burn(to,amount);
    }


}
```

2. MockWETH9.sol

```
// Copyright (C) 2015, 2016, 2017 Dapphub

// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.

pragma solidity =0.6.6;

contract WETH9 {
    string public name     = "Wrapped Ether";
    string public symbol   = "WETH";
    uint8  public decimals = 18;

    event  Approval(address indexed src, address indexed guy, uint wad);
    event  Transfer(address indexed src, address indexed dst, uint wad);
    event  Deposit(address indexed dst, uint wad);
    event  Withdrawal(address indexed src, uint wad);

    mapping (address => uint)                       public  balanceOf;
    mapping (address => mapping (address => uint))  public  allowance;

    function deposit() public payable {
        balanceOf[msg.sender] += msg.value;
        emit Deposit(msg.sender, msg.value);
    }
    function withdraw(uint wad) public {
        require(balanceOf[msg.sender] >= wad, "");
        balanceOf[msg.sender] -= wad;
        msg.sender.transfer(wad);
        emit Withdrawal(msg.sender, wad);
    }

    function totalSupply() public view returns (uint) {
        return address(this).balance;
    }

    function approve(address guy, uint wad) public returns (bool) {
        allowance[msg.sender][guy] = wad;
        emit Approval(msg.sender, guy, wad);
        return true;
    }

    function transfer(address dst, uint wad) public returns (bool) {
        return transferFrom(msg.sender, dst, wad);
    }

    function transferFrom(address src, address dst, uint wad)
        public
        returns (bool)
    {
        require(balanceOf[src] >= wad, "");

        if (src != msg.sender && allowance[src][msg.sender] != uint(-1)) {
            require(allowance[src][msg.sender] >= wad, "");
            allowance[src][msg.sender] -= wad;
        }

        balanceOf[src] -= wad;
        balanceOf[dst] += wad;

        emit Transfer(src, dst, wad);

        return true;
    }
}
```

3. Zebra.t.js

```
const {
    loadFixture,
} = require("@nomicfoundation/hardhat-network-helpers");

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

describe("Zebra swap unit test", function() {
    async function deployFixture() {
        const [Owner,Alice,Bob,...Users] = await ethers.getSigners();
        const WETH9 = await ethers.getContractFactory("WETH9");
        const weth = await WETH9.deploy();
        const MockERC20 = await ethers.getContractFactory("MockERC20");
        const tokenA = await
MockERC20.deploy("tokenA","TAT",ethers.parseEther("10000000"));
        const tokenB = await
MockERC20.deploy("tokenB","TBT",ethers.parseEther("10000000"));

        const ZebraFactory = await ethers.getContractFactory("ZebraFactory");
        const factory = await ZebraFactory.deploy(Owner.address);
        const ZebraRouter = await ethers.getContractFactory("ZebraRouter");
        const router = await ZebraRouter.deploy(factory.target,weth.target);
        await factory.setFeeToRate(6);
        await tokenA.approve(router.target,ethers.MaxUint256);
        await tokenB.approve(router.target,ethers.MaxUint256);
        tokenA.connect(Alice).approve(router.target,ethers.MaxUint256);
        tokenB.connect(Alice).approve(router.target,ethers.MaxUint256);


        return {
            factory,router,tokenA,tokenB,weth,Owner,Alice,Bob,Users,MockERC20
        };
    }

    it("add liquid tokenA and tokenB", async () => {
        const {factory,tokenA,tokenB,router,Owner,Alice,MockERC20} = await
loadFixture(deployFixture);
        let pair = await factory.getPair(tokenA.target,tokenB.target);
        expect(pair).eq(ethers.ZeroAddress);
        let e_pair = await factory.pairFor(tokenA.target,tokenB.target);
        await router.addLiquidity(
            tokenA.target,
            tokenB.target,
            ethers.parseEther("900"),
            ethers.parseEther("100"),
            1,
            1,
            Owner.address,
            9876543210
        );
        pair = await factory.getPair(tokenA.target,tokenB.target);
        expect(pair).eq(e_pair);
        let lp = MockERC20.attach(pair);
        let lp_balance = await lp.balanceOf(Owner.address);
        let origin = ethers.parseEther("300") - ethers.getBigInt(1000);
        expect(lp_balance).eq(origin);

        await tokenA.mint(Alice.address,ethers.parseEther("900"));
        await tokenB.mint(Alice.address,ethers.parseEther("100"));
        // add again should be successful
        await router.connect(Alice).addLiquidity(
            tokenA.target,
            tokenB.target,
            ethers.parseEther("900"),
            ethers.parseEther("100"),
            1,
            1,
            Alice.address,
            9876543210
        );

        lp_balance = await lp.balanceOf(Alice.address);
        await lp.connect(Alice).approve(router.target,ethers.MaxUint256);
        await router.connect(Alice).removeLiquidity(
            tokenA.target,
            tokenB.target,
            lp_balance,
            1,
            1,
            Alice.address,
            9876543210
        );
        expect(await tokenA.balanceOf(Alice)).eq(ethers.parseEther("900"));
        expect(await tokenB.balanceOf(Alice)).eq(ethers.parseEther("100"));
    });

    it("add liquid tokenA and ETH", async () => {
        const {factory,tokenA,weth,router,Owner,MockERC20} = await
loadFixture(deployFixture);
        let pair = await factory.getPair(tokenA.target,weth.target);
        expect(pair).eq(ethers.ZeroAddress);
        let e_pair = await factory.pairFor(tokenA.target,weth.target);
        await router.addLiquidityETH(
            tokenA.target,
            ethers.parseEther("900"),
            1,
            1,
            Owner.address,
            9876543210,
            {value:ethers.parseEther("100")}
        );
        pair = await factory.getPair(tokenA.target,weth.target);
        expect(pair).eq(e_pair);
        let lp = MockERC20.attach(pair);
        let lp_balance = await lp.balanceOf(Owner.address);
        let origin = ethers.parseEther("300") - ethers.getBigInt(1000);
        expect(lp_balance).eq(origin);
    });

    it("Swap tokenA and TokenB test", async () => {
        const {factory,tokenA,tokenB,router,Owner,Alice} = await
loadFixture(deployFixture);
        await router.addLiquidity(
            tokenA.target,
            tokenB.target,
            ethers.parseEther("900"),
            ethers.parseEther("100"),
            1,
            1,
            Owner.address,
            9876543210
        );
        let [reserveA, reserveB] = await
factory.getReserves(tokenA.target,tokenB.target);
        let amountIn = ethers.parseEther("2.0"); // tokenA
        let amountOut = await factory.getAmountOut(
            amountIn,
            reserveA,
            reserveB
        );
        await tokenA.mint(Alice.address,amountIn);
        await router.swapExactTokensForTokens(amountIn,1,
[tokenA.target,tokenB.target],Alice.address,9876543210);
        let balance = await tokenB.balanceOf(Alice.address);
        expect(balance).eq(amountOut);
    });

    it("Swap ETH and TokenA", async () => {
        const {factory,tokenA,weth,router,Owner,Alice} = await
loadFixture(deployFixture);
        await router.addLiquidityETH(
            tokenA.target,
            ethers.parseEther("900"),
            1,
            1,
            Owner.address,
            9876543210,
            {value:ethers.parseEther("100")}
        );

        let [reserveIn, reserveOut] = await
factory.getReserves(weth.target,tokenA.target);
        let amountIn = ethers.parseEther("2.0"); // weth
        let amountOut = await factory.getAmountOut(
            amountIn,
            reserveIn,
            reserveOut
        );
        await router.swapExactETHForTokens(
            1,
            [weth.target,tokenA.target],
            Alice.address,
            9876543210,
            {value:amountIn}
        );
        let balance = await tokenA.balanceOf(Alice.address);
        expect(balance).eq(amountOut);

        ([reserveIn, reserveOut] = await
factory.getReserves(tokenA.target,weth.target));
        amountOut = await factory.getAmountOut(
            balance,
            reserveIn,
            reserveOut
        );
        let lucky = ethers.getAddress("0x" + "0".repeat(36) + "1234");
        await router.swapExactTokensForETH(
            balance,
            1,
            [tokenA.target,weth.target],
            lucky,
            9876543210
        );
        balance = await ethers.provider.getBalance(lucky);
        expect(balance).eq(amountOut);
    });

});
```

4. UnitTestOutput

```
  Zebra swap unit test
    ✓ add liquid tokenA and tokenB (2119ms)
    ✓ add liquid tokenA and ETH (95ms)
    ✓ Swap tokenA and TokenB test (152ms)
    ✓ Swap ETH and TokenA (208ms)


   4 passing (3s)
```

11.2 External Functions Check Points

1. ZebraFactory_output.md

contract: ZebraFactory is IZebraFactory

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

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
1	allPairsLength()	view					
2	createPair(address,address)				Yes		
3	setFeeTo(address)						Only feeToSetter
·	setFeeToSetter(address)						Only feeToSetter
5	setFeeToRate(uint256)						Only feeToSetter
6	sortTokens(address,address)	pure					
7	pairFor(address,address)	view					
8	getReserves(address,address)	view					
9	quote(uint,uint,uint)	pure					
10	getAmountOut(uint,uint,uint)	view					
11	getAmountIn(uint,uint,uint)	view					
12	getAmountsOut(uint,address[])	view					
13	getAmountsIn(uint,address[])	view					

2. ZebraPair_output.md

contract: ZebraPair is IZebraPair, ZebraERC20

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

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
1	getReserves()	view					
2	initialize(address,address)						Only once
3	mint(address)		`lock`		Yes		
4	burn(address)		`lock`		Yes		
5	swap(uint,uint,address,bytes)		`lock`		Yes		
6	skim(address)		`lock`		Yes		
7	sync()		`lock`		Yes		
8	price(address,uint256)	view					
9	approve(address,uint)				Yes		
10	transfer(address,uint)				Yes		
11	transferFrom(address,address,uint)				Yes		
12	permit(address,address,uint,uint,uint8,bytes32,bytes32)				Yes		

3. ZebraRouter_output.md

contract: ZebraRouter is IZebraRouter, Ownable

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

Index	Function	StateMutability	Modifier	Param Check	IsUserInterface	Unit Test	Miscellaneous
1	receive()	payable					Only WETH
2	pairFor(address,address)	view					
3	setSwapMining(address)		`onlyOwner`				
4	addLiquidity(address,address,uint,uint,uint,uint,address,uint)		`ensure(deadline)`		Yes		
5	addLiquidityETH(address,uint,uint,uint,address,uint)	payable	`ensure(deadline)`		Yes		
6	removeLiquidity(address,address,uint,uint,uint,address,uint)		`ensure(deadline)`		Yes		
7	removeLiquidityETH(address,uint,uint,uint,address,uint)		`ensure(deadline)`		Yes		
8	removeLiquidityWithPermit(address,address,uint,uint,uint,address,uint,bool,uint8,bytes32,bytes32)				Yes		
9	removeLiquidityETHWithPermit(address,uint,uint,uint,address,uint,bool,uint8,bytes32,bytes32)				Yes		
10	removeLiquidityETHSupportingFeeOnTransferTokens(address,uint,uint,uint,address,uint)		`ensure(deadline)`		Yes		
11	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(address,uint,uint,uint,address,uint,bool,uint8,bytes32,bytes32)				Yes		
12	swapExactTokensForTokens(uint,uint,address[],address,uint)		`ensure(deadline)`		Yes		
13	swapTokensForExactTokens(uint,uint,address[],address,uint)		`ensure(deadline)`		Yes		
14	swapExactETHForTokens(uint,address[],address,uint)	payable	`ensure(deadline)`		Yes		
15	swapTokensForExactETH(uint,uint,address[],address,uint)		`ensure(deadline)`		Yes		
16	swapExactTokensForETH(uint,uint,address[],address,uint)		`ensure(deadline)`		Yes		
17	swapETHForExactTokens(uint,address[],address,uint)	payable	`ensure(deadline)`		Yes		
18	swapExactTokensForTokensSupportingFeeOnTransferTokens(uint,uint,address[],address,uint)		`ensure(deadline)`		Yes		
19	swapExactETHForTokensSupportingFeeOnTransferTokens(uint,address[],address,uint)	payable	`ensure(deadline)`		Yes		
20	swapExactTokensForETHSupportingFeeOnTransferTokens(uint,uint,address[],address,uint)		`ensure(deadline)`		Yes		
21	quote(uint256,uint256,uint256)	view					
22	getAmountOut(uint256,uint256,uint256)	view					
23	getAmountIn(uint256,uint256,uint256)	view					
24	getAmountsOut(uint256,address[])	view					
25	getAmountsIn(uint256,address[])	view					

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