BRO Token

AUDIT REPORT

Version 1.0.0

Serial No. 2024122400012020

Presented by Fairyproof

December 25, 2024

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the BRO TokenIssuance project.

Audit Start Time:

December 23, 2024

Audit End Time:

December 24, 2024

Audited Code's Github Repository:

https://github.com/breadNbutter42/BRO/blob/main/BRO/BroTokenWithPresale.sol

Audited Code's Github Commit Number When Audit Started:

073f35103199e533d0873da89805e18172fd01e9

Audited Code's Github Commit Number When Audit Ended:

073f35103199e533d0873da89805e18172fd01e9

Audited Source Files:

The source files audited include all the files as follows:

```
BRO/BroTokenWithPresale.sol
```

The goal of this audit is to review BRO's solidity implementation for its TokenIssuance 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 BRO team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Fairyproof

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.

Fairyproof

— 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.mybro.xyz/

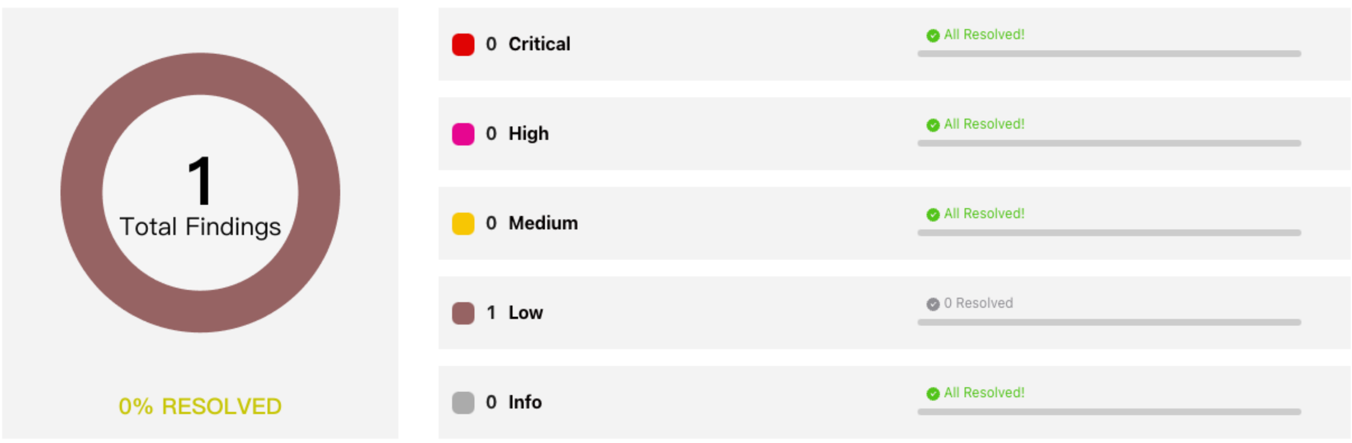
Source Code: https://github.com/breadNbutter42/BRO/blob/main/BRO/BroTokenWithPresale.sol

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

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2024122400012020	Fairyproof Security Team	Dec 23, 2024 - Dec 24, 2024	Low Risk

Fairyproof



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 was uncovered. The BRO team acknowledged 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 BRO

BRO is a community token, with no intrinsic value or expectation of financial return, and no formal team or roadmap.

The above description is quoted from relevant documents of BRO.

04. Major functions of audited code

Overview

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

- Blockchain: Avalanche
- Token Standard: ERC20
- Token Name: My Bro

Fairyproof

- Token Symbol: BRO
- Decimals: 18
- Current Supply: 420,690,000,000,000
- Max Supply: 420,690,000,000,000

Key Features

1.**Phased Token Launch**:

- Presale phase
- LP seeding phase
- Token distribution phase
- Whitelist trading phase
- Public trading phase

2. **Presale Mechanism**:

- Distribution based on AVAX contribution
- Emergency withdrawal before LP seeding
- Protection against user funds being locked

3. **Permanent Liquidity Protection**:

- Initial LP tokens are burned
- Ensures permanent liquidity in the pool
- Prevents rug pulls through LP removal

4. **Trading Restrictions**:

- Phase-based trading controls
- Whitelist phase protection for presale participants
- Public trading phase has no limit

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

06. Severity level reference

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

Fairyproof

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.

Fairyproof

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 didn't find issues or risks in these functions or areas at the time of writing.

Fairyproof

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Liquidity Removal Restriction in Whitelist Phase	Implementation Vulnerability	Low	Acknowledged

09. Issue descriptions

[FP-1] Liquidity Removal Restriction in Whitelist Phase

Implementation Vulnerability Low Acknowledged

Issue/Risk: Implementation Vulnerability

Description:

During the whitelist phase (Phase 3), users can add liquidity but cannot remove it. This is because when removing liquidity, tokens are first transferred to the router contract, which is subject to whitelist transfer restrictions.
Current Implementation:

```
if (idoBuyer != LFJ_V1_PAIR_ADDRESS) {
  userTokensFromWL[idoBuyer] += _amountTokensToTransfer;
  // ... transfer restrictions
}
```

Recommendation:

Add the router address to the exemption list.

Update:

The Bro team is aware of this issue, but since the whitelist trading phase only lasts for 5 minutes, users can remove liquidity after this brief period. Meanwhile, the team will inform users about this situation in advance to prevent any unintended operations.

Status:

The Bro team is aware of this 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.

- Missing Presale Buyers Length Function
  The contract lacks a direct way to query the total number of presale buyers, which would be useful for frontend integration and progress tracking.

  Recommended Addition:

```
/// @dev Returns the total number of presale buyers
function presaleBuyersLength() public view returns (uint256) {
  return presaleBuyers.length;
}
```

Fairyproof

11. Appendices

11.1 Unit Test

1. Bro.t.sol

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;
```

```
import {Test, console} from "forge-std/Test.sol";
import {BroTokenWithPresale,ITJUniswapV2Router01,ERC20} from "../src/Bro.sol";

contract BroTest is Test {
    address public Alice = makeAddr("Alice");
    address public Bob = makeAddr("Bob");
    address public Charlie = makeAddr("Charlie");
    address public Dave = makeAddr("Dave");
    address public Eve = makeAddr("Eve");
    address public Frank = makeAddr("Frank");
    address public Grace = makeAddr("Grace");

    BroTokenWithPresale public broToken;
    ITJUniswapV2Router01 public router;

    // event
    event AirdropCompleted(
        address indexed caller
    );

    event LPSeeded(
        address indexed caller,
        uint256 avaxAmount,
        uint256 broAmount
    );

    event BuyerAdded(
        address indexed buyer
    );

    event PresaleBought(
        address indexed buyer,
        uint256 amount
    );

    event AvaxWithdraw(
        address indexed to,
        uint256 amount
    );

    event TokensClaimed(
        address indexed to,
        uint256 amount
    );

    event DustSent(
        address indexed to,
        uint256 amount
    );

    //  We're using fork testing, so we don't need to set block height and time in setUp
    function setUp() public {
```

Fairyproof

```
        vm.deal(Alice, 10 ether);
        vm.deal(Bob, 10 ether);
        vm.deal(Charlie, 10 ether);
        vm.deal(Dave, 10 ether);
        vm.deal(Eve, 10 ether);
        vm.deal(Frank, 10 ether);
        vm.deal(Grace, 10 ether);
        router =
ITJUniswapV2Router01(address(0x60aE616a2155Ee3d9A68541Ba4544862310933d4));
        broToken = new BroTokenWithPresale();
    }

    function test_meta() public view{
        assertEq(broToken.name(), "My Bro");
        assertEq(broToken.symbol(), "BRO");
        assertEq(broToken.totalSupply(), 420690 * (10**9) * (10**18));
        assertEq(broToken.balanceOf(address(broToken)), 420690 * (10**9) * (10**18));
        assertEq(broToken.PRESALERS_BRO_SUPPLY(), 210345 * (10**9) * (10**18));
        assertEq(broToken.LP_BRO_SUPPLY(), 210345 * (10**9) * (10**18));
    }

    function test_constants() public view{
        assertEq(broToken.IDO_START_TIME(), 1735554420); //IDO start time
        assertEq(broToken.PRESALE_END_TIME(), 1735554420 - 2 * 3600); //presale end time
(IDO start time - 2 hours)
        assertEq(broToken.SEEDING_TIME(), 1735554420 - 2 * 3600 + 1 * 60); // seed start
time, one minute after presale end time
        assertEq(broToken.WL_END_TIME(), 1735554420 + 5 * 60); //whitelist end time
    }

    function test_initial_state() public view{
        assertEq(broToken.tradingPhase(), 0);
        assertEq(broToken.tradingActive(), false);
        assertEq(broToken.tradingRestricted(), false);
        assertEq(broToken.lpSeeded(), false);
        assertEq(broToken.airdropCompleted(), false);
        assertEq(broToken.airdropIndex(), 0);
        assertEq(broToken.totalAvaxPresale(), 0);
        assertEq(broToken.previousBuyer(Alice), false);
        assertEq(broToken.airdropSlot(Alice), 0);
    }

    function test_presale_failed() public{
        vm.prank(Alice);
        vm.expectRevert("Minimum buy of 1 AVAX per transaction; Not enough AVAX sent");
        broToken.buyPresale{value: 0.9 ether}();

        vm.warp(broToken.PRESALE_END_TIME() + 1);
        vm.expectRevert("Presale has already ended");
        broToken.buyPresale{value: 1 ether}();
    }
```

Fairyproof

```
    /// Test presale, it has two methods: calling function and sending AVAX directly
    function test_presale() public{
        assertEq(broToken.tradingPhase(), 0);

        // Check initial values
        assertEq(broToken.previousBuyer(Alice), false);
        assertEq(broToken.totalAvaxUserSent(Alice), 0);
        assertEq(broToken.totalAvaxPresale(), 0);
        // Note that the initial value of airdropSlot is 0
        assertEq(broToken.airdropSlot(Alice), 0);
        // Will trigger event
        vm.expectEmit(true, false, false, false);
        emit BuyerAdded(Alice);

        // Call function
        vm.prank(Alice);
        broToken.buyPresale{value: 1 ether}();
        //accesses
        assertEq(broToken.previousBuyer(Alice), true);
        assertEq(broToken.totalAvaxUserSent(Alice), 1 ether);
        assertEq(broToken.totalAvaxPresale(), 1 ether);
        assertEq(broToken.airdropSlot(Alice), 0);
        // Send AVAX directly
        vm.deal(Alice, 1 ether);
        vm.prank(Alice);
        vm.expectEmit(true, false, false, true);
        emit PresaleBought(Alice, 1 ether);
        (bool success, ) = address(broToken).call{value: 1 ether}("");
        assertEq(success, true);

        // Check values
        assertEq(broToken.previousBuyer(Alice), true);
        assertEq(broToken.totalAvaxUserSent(Alice), 2 ether);
        assertEq(broToken.totalAvaxPresale(), 2 ether);
        assertEq(broToken.airdropSlot(Alice), 0);

        // bob buy 1.1 ether
        vm.expectEmit(true, false, false, false);
        emit BuyerAdded(Bob);
        vm.prank(Bob);
        broToken.buyPresale{value: 1.1 ether}();
        // Check values
        assertEq(broToken.previousBuyer(Bob), true);
        assertEq(broToken.totalAvaxUserSent(Bob), 1.1 ether);
        assertEq(broToken.totalAvaxPresale(), 3.1 ether);
        assertEq(broToken.airdropSlot(Bob), 1);
    }

    /// Test emergencyWithdraw failed
    function test_emergencyWithdraw_failed() public{
        // Scenario 1: Not purchased
        vm.prank(Alice);
```

Fairyproof

```
        vm.expectRevert("No AVAX to withdraw");
        broToken.emergencyWithdraw();
        // Scenario 2: Already exited
        vm.prank(Alice);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Alice);
        broToken.emergencyWithdraw();
        vm.prank(Alice);
        vm.expectRevert("No AVAX to withdraw");
        broToken.emergencyWithdraw();
        // Scenario 3: Already seeded
        vm.prank(Alice);
        broToken.buyPresale{value: 1 ether}();
        vm.warp(broToken.SEEDING_TIME() + 1);
        broToken.seedLP();
        vm.prank(Alice);
        vm.expectRevert("LP has already been seeded");
        broToken.emergencyWithdraw();
    }

    function test_emergencyWithdraw() public{
        vm.prank(Alice);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Bob);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Charlie);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Dave);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Eve);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Frank);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Grace);
        broToken.buyPresale{value: 1 ether}();
        // Check values
        assertEq(broToken.totalAvaxPresale(), 7 ether);
        assertEq(broToken.previousBuyer(Dave), true);
        assertEq(broToken.totalAvaxUserSent(Dave), 1 ether);
        assertEq(broToken.airdropSlot(Dave), 3);
        // Someone exits in the middle
        vm.prank(Dave);
        vm.expectEmit(true, false, false, true);
        emit AvaxWithdraw(Dave, 1 ether);
        broToken.emergencyWithdraw();
        // Check values
        assertEq(broToken.totalAvaxPresale(), 6 ether);
        assertEq(broToken.previousBuyer(Dave), false);
        assertEq(broToken.totalAvaxUserSent(Dave), 0);
        assertEq(broToken.airdropSlot(Dave), 3);  // Not updated
        assertEq(broToken.airdropSlot(Grace), 3);  // Moved to the middle
        assertEq(broToken.airdropSlot(Eve), 4);
```

Fairyproof

```
        assertEq(broToken.airdropSlot(Frank), 5);

        // Last person exits
        vm.prank(Frank);
        broToken.emergencyWithdraw();
        // Check values
        assertEq(broToken.totalAvaxPresale(), 5 ether);
        assertEq(broToken.previousBuyer(Frank), false);
        assertEq(broToken.totalAvaxUserSent(Frank), 0);
        assertEq(broToken.airdropSlot(Dave), 3);  // Not updated
        assertEq(broToken.airdropSlot(Grace), 3);
        assertEq(broToken.airdropSlot(Eve), 4);
        assertEq(broToken.airdropSlot(Frank), 5);  // Not updated

        // David joins again
        vm.prank(Dave);
        broToken.buyPresale{value: 1 ether}();
        // Check values
        assertEq(broToken.totalAvaxPresale(), 6 ether);
        assertEq(broToken.airdropSlot(Grace), 3);
        assertEq(broToken.airdropSlot(Eve), 4);
        assertEq(broToken.airdropSlot(Frank), 5);  // Not updated
        assertEq(broToken.airdropSlot(Dave), 5);  // Added at the end, overwrote the
previous 3
    }

    function test_seedLP_failed() public{
        vm.prank(Alice);
        broToken.buyPresale{value: 1 ether}();
        vm.expectRevert("Presale time plus buffer has not yet ended");
        broToken.seedLP();

        vm.warp(broToken.SEEDING_TIME() + 1);
        broToken.seedLP();

        vm.expectRevert("LP has already been seeded");
        broToken.seedLP();
    }

    function test_seedLP_and_airdrop() public{
        assertEq(broToken.lpSeeded(), false);
        assertEq(broToken.tradingPhase(),0,"should be presale");
        vm.prank(Alice);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Bob);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Charlie);
        broToken.buyPresale{value: 1.2 ether}();
        vm.warp(broToken.SEEDING_TIME() + 1);
        assertEq(broToken.tradingPhase(),1,"should be seeding");
        assertEq(broToken.balanceOf(address(broToken)), 210345 * 2 * (10**9) * (10**18));
        vm.expectEmit(true, false, false, true);
```

Fairyproof

```
        emit LPSeeded(address(this),3.2 ether, 210345 * (10**9) * (10**18));
        broToken.seedLP();
        // Check values
        assertEq(broToken.lpSeeded(), true);
        assertEq(broToken.totalAvaxPresale(), 3.2 ether);
        assertEq(broToken.totalAvaxUserSent(Alice), 1 ether);
        assertEq(broToken.totalAvaxUserSent(Bob), 1 ether);
        assertEq(broToken.totalAvaxUserSent(Charlie), 1.2 ether);
        assertEq(broToken.balanceOf(address(broToken)), 210345 * (10**9) * (10**18));
        assertEq(broToken.tradingPhase(),2,"should be token dispersal");

        // airdrop
        broToken.airdropBuyers(2);
        assertEq(broToken.balanceOf(Alice), 210345 * (10**9) * (10**18) * 1 ether / 3.2
ether);
        assertEq(broToken.balanceOf(Bob), 210345 * (10**9) * (10**18) * 1 ether / 3.2
ether);
        uint256 expected = 210345 * (10**9) * (10**18) * 1.2 ether / 3.2 ether;
        uint256 dust = broToken.PRESALERS_BRO_SUPPLY() - broToken.balanceOf(Alice) -
broToken.balanceOf(Bob) - expected;
        vm.expectEmit(true, false, false, true);
        emit DustSent(Charlie, dust);
        broToken.airdropBuyers(2);
        assertEq(broToken.balanceOf(Charlie), broToken.PRESALERS_BRO_SUPPLY() -
broToken.balanceOf(Alice) - broToken.balanceOf(Bob));
        assertEq(broToken.airdropCompleted(), true);

        vm.expectRevert("Airdrop has already been completed");
        broToken.airdropBuyers(2);

    }

    function test_MixedAirdropAndClaim() public{
        vm.prank(Alice);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Bob);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Charlie);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Dave);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Eve);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Frank);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Grace);
        broToken.buyPresale{value: 1 ether}();

        vm.warp(broToken.SEEDING_TIME() + 1);
        broToken.seedLP();
```

Fairyproof

```
        broToken.claimTokens(Bob);
        broToken.airdropBuyers(2);
        broToken.airdropBuyers(2);
        // Eve and Frank are not claimed yet
        assertEq(broToken.balanceOf(Eve), 0);
        assertEq(broToken.balanceOf(Frank), 0);
        broToken.claimTokens(Eve);
        vm.expectEmit(true, false, false, false);
        emit AirdropCompleted(Frank);
        broToken.claimTokens(Frank);
    }

    function test_tradingActive() public view{
        assertEq(broToken.tradingActive(), false);
    }

    function test_EmergencyWithdrawStateCleanup() public {
        address BUYER777 = makeAddr("BUYER777");
        address BUYER888 = makeAddr("BUYER888");
        address BUYER999 = makeAddr("BUYER999");
        BroTokenWithPresale bro = broToken;
        // 1. Set multiple buyers
        vm.deal(BUYER777, 2 ether);
        vm.deal(BUYER888, 2 ether);
        vm.deal(BUYER999, 2 ether);

        // BUYER777 buys
        vm.prank(BUYER777);
        (bool success,) = address(bro).call{value: 1 ether}("");
        require(success, "Buy failed");

        // BUYER888 buys
        vm.prank(BUYER888);
        (success,) = address(bro).call{value: 1 ether}("");
        require(success, "Buy failed");

        // BUYER999 buys
        vm.prank(BUYER999);
        (success,) = address(bro).call{value: 1 ether}("");
        require(success, "Buy failed");

        // 2. Record initial state
        assertEq(bro.totalAvaxPresale(), 3 ether, "Initial total should be 3 AVAX");
        assertEq(bro.presaleBuyers(0), BUYER777, "First buyer should be BUYER777");
        assertEq(bro.presaleBuyers(1), BUYER888, "Second buyer should be BUYER888");
        assertEq(bro.presaleBuyers(2), BUYER999, "Third buyer should be BUYER999");

        // 3. BUYER888 (middle user) exits
        uint256 initialBalance = BUYER888.balance;
        vm.prank(BUYER888);
        bro.emergencyWithdraw();
```

Fairyproof

```
        // 4. Verify state cleanup
        // 4.1 Verify AVAX refund
        assertEq(BUYER888.balance, initialBalance + 1 ether, "AVAX should be refunded");

        // 4.2 Verify total update
        assertEq(bro.totalAvaxPresale(), 2 ether, "Total should be reduced");

        // 4.3 Verify user state cleanup
        assertEq(bro.totalAvaxUserSent(BUYER888), 0, "User AVAX sent should be reset");
        assertFalse(bro.previousBuyer(BUYER888), "Previous buyer flag should be reset");

        // 4.4 Verify array update
        assertEq(bro.presaleBuyers(0), BUYER777, "First buyer should still be BUYER777");
        assertEq(bro.presaleBuyers(1), BUYER999, "BUYER999 should be moved to BUYER888's
position");

        // 4.5 Verify airdropSlot update
        assertEq(bro.airdropSlot(BUYER999), 1, "BUYER999's slot should be updated");

        // 5. Verify that the exiting user can re-purchase
        vm.prank(BUYER888);
        (success,) = address(bro).call{value: 2 ether}("");
        require(success, "Buy failed");

        // Verify state after re-purchase
        assertTrue(bro.previousBuyer(BUYER888), "Should be marked as buyer again");
        assertEq(bro.totalAvaxUserSent(BUYER888), 2 ether, "New purchase amount should be
recorded");
        assertEq(bro.totalAvaxPresale(), 4 ether, "Total should include new purchase");
    }


    function test_tradingPhase() public {
        // Initially in presale phase
        assertEq(broToken.tradingPhase(), 0);
        // Buy presale
        vm.prank(Alice);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Bob);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Charlie);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Dave);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Eve);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Frank);
        broToken.buyPresale{value: 1 ether}();
        vm.prank(Grace);
        broToken.buyPresale{value: 1 ether}();

        // Presale phase ends, start seeding
```

Fairyproof

```
        vm.warp(broToken.SEEDING_TIME() + 1);
        assertEq(broToken.tradingPhase(), 1);
        // Cannot buy now
        vm.prank(Alice);
        vm.expectRevert("Presale has already ended");
        broToken.buyPresale{value: 1 ether}();
        // Add lp, now in airdrop phase
        broToken.seedLP();
        assertEq(broToken.tradingPhase(), 2);

        broToken.airdropBuyers(2);
        broToken.airdropBuyers(2);

        assertEq(broToken.tradingActive(), false);
        assertEq(broToken.tradingRestricted(), false);

        // Time enters IDO phase
        vm.warp(broToken.IDO_START_TIME() + 1);
        assertEq(broToken.tradingPhase(), 3);
        assertEq(broToken.tradingActive(), true);
        assertEq(broToken.tradingRestricted(), true);

        // Need to test whether it is difficult to claim airdrop and whether it can be
purchased, adding/removing liquidity and transferring
        // Can claim airdrop (as long as the last person has not claimed, it can still be
claimed, but after claiming, it cannot be purchased again)
        broToken.airdropBuyers(2);
        // alice adds liquidity
        vm.prank(Alice);
        broToken.approve(address(router), 2 ether);
        vm.prank(Alice);
        router.addLiquidityAVAX{value: 0.1 ether}(address(broToken), 2 ether, 0, 0,
address(Alice), block.timestamp);
        address pair = broToken.LFJ_V1_PAIR_ADDRESS();
        uint lp = ERC20(pair).balanceOf(address(Alice));
        // Alice removes liquidity, this fails because tokens go to router contract first,
and router's limit is 0
        vm.prank(Alice);
        ERC20(pair).approve(address(router), lp);
        vm.prank(Alice);
        vm.expectRevert("Joe: TRANSFER_FAILED");
        router.removeLiquidityAVAX(address(broToken), lp, 0, 0, address(Alice),
block.timestamp);

        // Bob buys
        vm.prank(Bob);
        address[] memory path = new address[](2);
        path[0] = broToken.WAVAX_ADDRESS();
        path[1] = address(broToken);
        uint256[] memory amounts = router.swapExactAVAXForTokens{value: 1 ether}(0, path,
address(Bob), block.timestamp);
        uint amountOut = amounts[1];
```

```
        assertEq(broToken.userTokensFromWL(Bob), amountOut);
        uint limit = broToken.presaleTokensPurchased(Bob);
        assert(limit > amountOut);
        // Large purchase fails
        vm.deal(Bob, 1000 ether);
        vm.prank(Bob);
        vm.expectRevert("Joe: TRANSFER_FAILED");
        router.swapExactAVAXForTokens{value: 1000 ether}(0, path, address(Bob),
block.timestamp);

        // Eve can also buy, because claiming airdrop does not count towards the limit
        assertEq(broToken.userTokensFromWL(Eve), 0);
        vm.prank(Eve);
        amounts = router.swapExactAVAXForTokens{value: 1 ether}(0, path, address(Eve),
block.timestamp);
        amountOut = amounts[1];
        assertEq(broToken.userTokensFromWL(Eve), amountOut);

        // Cannot transfer to others
        address new_user = makeAddr("new_user");
        vm.prank(Eve);
        vm.expectRevert("Cannot receive more tokens than purchased in the presale during
WL phase");
        broToken.transfer(new_user, amountOut);

        vm.deal(new_user, 1 ether);
        // Cannot buy
        vm.prank(new_user);
        vm.expectRevert("Joe: TRANSFER_FAILED");
        router.swapExactAVAXForTokens{value: 1 ether}(0, path, address(new_user),
block.timestamp);

        // IDO phase ends
        vm.warp(broToken.WL_END_TIME() + 1);
        assertEq(broToken.tradingPhase(), 4);
        assertEq(broToken.tradingRestricted(), false);
        assertEq(broToken.tradingActive(), true);
        // Can transfer to others
        vm.prank(Eve);
        broToken.transfer(new_user, amountOut);

        // Can buy
        vm.prank(new_user);
        router.swapExactAVAXForTokens{value: 1 ether}(0, path, address(new_user),
block.timestamp);

        // Large purchase
        vm.deal(Bob, 1000 ether);
        vm.prank(Bob);
        router.swapExactAVAXForTokens{value: 1000 ether}(0, path, address(Bob),
block.timestamp);
```

Fairyproof

```
        // Claim airdrop
        broToken.claimTokens(Grace);
        assertEq(broToken.balanceOf(Grace) > 0, true);
    }
}
```

2. UnitTestOutPut

```
[⠊] Compiling...
[⠊] Compiling 1 files with Solc 0.8.28
[⠒] Solc 0.8.28 finished in 1.00s
Compiler run successful!

Ran 13 tests for test/Bro.t.sol:BroTest
[PASS] test_EmergencyWithdrawStateCleanup() (gas: 390448)
[PASS] test_MixedAirdropAndClaim() (gas: 1363454)
[PASS] test_constants() (gas: 12477)
[PASS] test_emergencyWithdraw() (gas: 772918)
[PASS] test_emergencyWithdraw_failed() (gas: 482268)
[PASS] test_initial_state() (gas: 32286)
[PASS] test_meta() (gas: 26606)
[PASS] test_presale() (gas: 284636)
[PASS] test_presale_failed() (gas: 41461)
[PASS] test_seedLP_and_airdrop() (gas: 834605)
[PASS] test_seedLP_failed() (gas: 417249)
[PASS] test_tradingActive() (gas: 10537)
[PASS] test_tradingPhase() (gas: 2044255)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 11.33ms (16.30ms CPU time)
```

Fairyproof

11.2 External Functions Check Points

1. Bro_check_point.md

File: src/Bro.sol

contract: BroTokenWithPresale is ERC20, ERC20Permit, ReentrancyGuard

(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			Yes	Passed	
2	seedLP()		`nonReentrant`, `afterPresale`, `notSeeded`		Yes	Passed	OnlyOnce
3	buyPresale()	payable			Yes	Passed	
4	emergencyWithdraw()		`nonReentrant`, `notSeeded`		Yes	Passed	NonReentrant
5	airdropBuyers(uint256)		`nonReentrant`, `afterPresale`, `seeded`		Yes	Passed	
6	claimTokens(address)		`nonReentrant`, `afterPresale`, `seeded`		Yes	Passed	
7	tradingActive()	view				Passed	
8	tradingRestricted()	view				Passed	
9	tradingPhase()	view				Passed	
10	presaleTokensPurchased(address)	view	`afterPresale`		Yes	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