BLOCKAPEX

SMART CONTRACT SECURITY

v 1.0

Date: 19-03-2025

Prepared For: RandomDEX Token Contract

SECURITY REPORT
BLOCKAPEX VERIFIED

About BlockApex

Founded in early 2021 BlockApex is a security-first blockchain consulting firm. We offer services in a wide range of areas including Audits for Smart Contracts, Blockchain Protocols, Tokenomics along with Invariant development (i.e., test-suite) and Decentralized Application Penetration Testing. With a dedicated team of over 40+ experts dispersed globally, BlockApex has contributed to enhancing the security of essential software components utilized by many users worldwide, including vital systems and technologies.

BlockApex has a focus on blockchain security, maintaining an expertise hub to navigate this dynamic field. We actively contribute to security research and openly share our findings with the community. Our work is available for review at our public repository, showcasing audit reports and insights into our innovative practices.

To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on Twitter and explore our GitHub. For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please visit our Contact page at our website , or reach out to us via email at hello@blockapex.io.

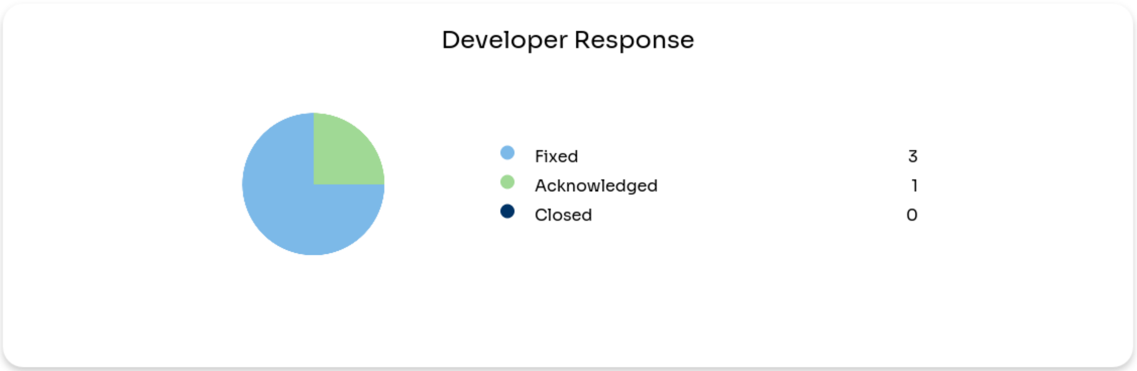
Contents

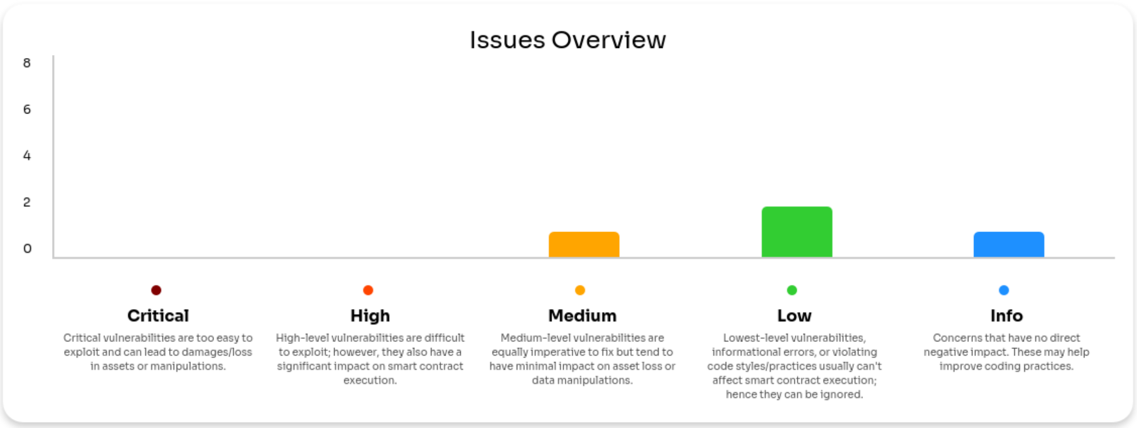
1 Executive Summary **4**
- 1.1 Scope . . . 5
  - 1.1.1 In Scope . . . 5
  - 1.1.2 Out of Scope . . . 5
- 1.2 Methodology . . . 5
- 1.3 Questions for Security Assessment . . . 6
- 1.4 Status Descriptions . . . 7
- 1.5 Summary of Findings Identified . . . 8

2 Findings and Risk Analysis **9**
- 2.1 Missing Fee Deduction in Swap Calculation Causes Transaction Reverts Due to Incorrect Output Amount . . . 9
- 2.2 Listing Timestamp Can Be Updated After Token is Listed . . . 11
- 2.3 Missing Validation for AntiBot Fees Can Cause Underflow in Fee Calculation . . . 12
- 2.4 Unnecessary receive() Function Allows Unintended ETH Transfers . . . 13

1 Executive Summary

The audit was conducted on the RandomDEX Token Contract, undertaken by an individual auditor. This comprehensive review employed a meticulous manual code analysis approach, focusing exclusively on a line-by-line examination of the contract's code to identify potential vulnerabilities, coding best practices deviations, and areas for optimization





1.1 Scope

1.1.1 In Scope

RandomDEX is a decentralized exchange (DEX) token contract that implements a fee-based mechanism for transactions. It allows fee collection, conversion to ETH, and distribution to a designated fee collector.

1. **Token Contract** – RandomDEX facilitates token trading with built-in fee structures for buy, sell, and anti-bot protection.
2. **Fee Collection and Conversion** – Accumulated fees from transactions are swapped for ETH and sent to a designated fee collector.
3. **Slippage and Fee Mechanisms** – The contract supports slippage tolerance settings to manage price impact and trading efficiency.

Contracts in scope:

1. Contracts/*

Initial Commit Hash: c65b023b81573388d2eb0748e909c984abf58877

Fixed Commit Hash: fa876aa3471b26600d14fd949e5c06b792af4f08

1.1.2 Out of Scope

All features or functionalities not delineated within the “In Scope” section of this document shall be deemed outside the purview of this audit.

1.2 Methodology

The audit of the RandomDEX Token Contract was conducted over the course of 7 days, utilizing a straightforward, manual code review approach by one auditor. The methodology began with a reconnaissance phase to gain an initial understanding of the contract’s structure and intended functionalities. Following this, the auditor engaged in a detailed, line-by-line examination of the code. This manual review process focused on identifying logical flaws, security vulnerabilities, and opportunities for optimization, while ensuring adherence to Solidity best practices, security design patterns, and coding standards for clarity and efficiency.

1.3 Questions for Security Assessment

1. Does the contract correctly account for transfer fees when estimating swap output amounts in DEX interactions?
2. How does the contract handle fee adjustments in getAmountsOut calculations to prevent reverts due to insufficient output amounts?
3. Does the contract properly validate the listing status before allowing timestamp updates?
4. How does the contract ensure that excessive anti-bot fees do not block token transfers?
5. Are administrative functions such as fee collection, slippage tolerance updates, and listing timestamp modifications protected by appropriate role-based access controls?
6. How does the contract handle the collection and conversion of transaction fees to ETH?
7. Are there any potential failure scenarios in the fee conversion mechanism, such as failing swaps due to inaccurate fee calculations?

1.4 Status Descriptions

Acknowledged: The issue has been recognized and is under review. It indicates that the relevant team is aware of the problem and is actively considering the next steps or solutions.

Fixed: The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

Closed: This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.

1.5 Summary of Findings Identified

S.NO	SEVERITY	FINDINGS	STATUS
#1	MEDIUM	Missing Fee Deduction in Swap Calculation Causes Transaction Reverts Due to Incorrect Output Amount	FIXED
#2	LOW	Listing Timestamp Can Be Updated After Token is Listed	ACKNOWLEDGED
#3	LOW	Missing Validation for AntiBot Fees Can Cause Underflow in Fee Calculation	FIXED
#4	INFO	Unnecessary receive() Function Allows Unintended ETH Transfers	FIXED

2 Findings and Risk Analysis

2.1 Missing Fee Deduction in Swap Calculation Causes Transaction Reverts Due to Incorrect Output Amount

Severity: Medium

Status: Fixed

Location :

1. `contracts/RandomDEX.sol`

Description The `_swapRDXForETH` function does not account for transfer fees when estimating the output ETH amount. The function uses `getAmountsOut(rdxAmount, path)[1]` to determine ethAmountExpected, but this calculation assumes the full rdxAmount is available for swapping. However, the token has a transfer fee, the actual amount received by the Uniswap pool is lower than rdxAmount, leading to an overestimated ethAmountExpected. Since minEthAmount is derived from this overestimated value, the transaction is likely to revert due to `INSUFFICIENT_OUTPUT_AMOUNT` when the actual ETH received is lower than minEthAmount.

Impact The fee conversion mechanism may frequently fail, preventing the admin from claiming fees in ETH.

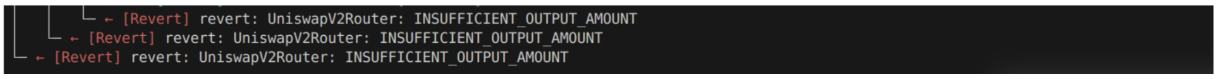
Proof of Concept

```
function setUp() public {
    RandomDEX.Fees memory anti_bot_fees;
    anti_bot_fees.buy = 2500; // 25% fee on transactions
    anti_bot_fees.sell = 2500;

    vm.startPrank(default_admin);
    RandomDEX_con.grantRole(RandomDEX_con.MINT_ROLE(), default_admin);
    RandomDEX_con.mint(user, 100 ether);
    RandomDEX_con.mint(default_admin, 1000 ether);
    vm.stopPrank();
}

function test_collecting_fees() public {
    vm.startPrank(user);
    RandomDEX_con.transfer(address(RandomDEX_con), 1 ether);
    // User sends 1 token as a fee, which is directly transferred to the contract.
    vm.stopPrank();

    vm.startPrank(default_admin);
    RandomDEX_con.updateSlippageTolerance(500); // Set slippage to 5%
    RandomDEX_con.claimFeeInEth();
    // This calls `_swapRDXForETH`, which fails due to incorrect expected ETH calculation.
    vm.stopPrank();
}
```



```
│   │   └─ ← [Revert] revert: UniswapV2Router: INSUFFICIENT_OUTPUT_AMOUNT
│   └─ ← [Revert] revert: UniswapV2Router: INSUFFICIENT_OUTPUT_AMOUNT
└─ ← [Revert] revert: UniswapV2Router: INSUFFICIENT_OUTPUT_AMOUNT
```

Figure 1: Test Output

Recommendation :

1. Adjust rdxAmount in getAmountsOut to reflect the actual amount that will be received by the Uniswap pool after transfer fees.
2. Exclude the token contract from transfer fees to ensure that the full rdxAmount is available for swapping without deductions.

2.2 Listing Timestamp Can Be Updated After Token is Listed

Severity: Low

Status: Acknowledged

Location :

1. `contracts/RandomDEX.sol`

Description The setListingTimestamp function is intended to prevent updates to listingTimestamp after the token is listed. The setListingTimestamp function does not verify if the token is listed on a DEX before allowing updates. If the token is listed before listingTimestamp is reached, the admin can still modify it, contradicting the intended restriction

For Example:

1. The admin sets listingTimestamp to T1.
2. The token is listed on a DEX before T1 (at T2 < T1).
3. Since block.timestamp is still below T1, the admin can call setListingTimestamp again to change listingTimestamp, even though the token is already listed

Recommendation Implement a check to ensure the token is not already listed before allowing updates

2.3 Missing Validation for AntiBot Fees Can Cause Underflow in Fee Calculation

Severity: Low

Status: Fixed

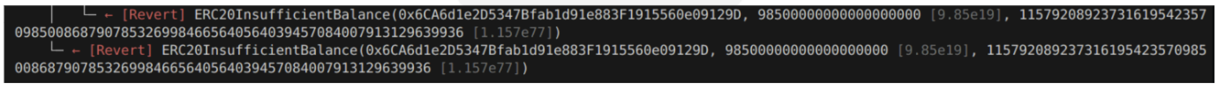
Location :

1. `contracts/ERC20Fee.sol`

Description The constructor of ERC20Fee correctly validates the fees_ struct to ensure that buy and sell fees do not exceed `maximumNumerator_`. However, the same validation is missing for `antiBotFees_`, allowing it to be set above `maximumNumerator_` or even above `denominator_`. Since fee deduction is calculated as:

```
fee = (value * buyFee) / denominator;
unchecked {
value -= fee;
}
```

If antiBotFees.buy or antiBotFees.sell is greater than denominator_, the calculated fee can be larger than value, causing an underflow when subtracting fee from value. This results in failed transfers, effectively preventing token movement when anti-bot fees are active.



```
    │    └─ ← [Revert] ERC20InsufficientBalance(0x6CA6d1e2D5347Bfab1d91e883F1915560e09129D, 98500000000000000000 [9.85e19], 115792089237316195423570985008687907853269984665640564039457084007913129639936 [1.157e77])
    └─ ← [Revert] ERC20InsufficientBalance(0x6CA6d1e2D5347Bfab1d91e883F1915560e09129D, 98500000000000000000 [9.85e19], 115792089237316195423570985008687907853269984665640564039457084007913129639936 [1.157e77])
```

Figure 2: Result

Recommendation :

1. Add validation in the constructor to ensure `antiBotFees_`.buy and `antiBotFees_`.sell do not exceed `maximumNumerator_`.

```
if (antiBotFees_.buy &gt; maximumNumerator_ || antiBotFees_.sell &gt; maximumNumerator_) {
    revert CannotBeBiggerThanMaximumNumerator();
}
```

2. Introduce a function to update `antiBotFees_` after deployment, with proper access control, to correct misconfigurations if needed.

2.4 Unnecessary receive() Function Allows Unintended ETH Transfers

Severity: Info

Status: Fixed

Location :

1. `contracts/RandomDEX.sol`

Description The `receive()` function allows the contract to accept ETH, but the contract does not require direct ETH deposits. This can lead to unintended ETH transfers and locked funds.

Recommendation Remove the `receive()` function to prevent accidental ETH deposits.