

# Security Audit

# Report for Mimboku Protocol

**Date:** May 26, 2025 **Version:** 1.0

**Contact:** [contact@blocksec.com](mailto:contact@blocksec.com)

# Contents

<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 About Target Contracts	1
1.2 Disclaimer	1
1.3 Procedure of Auditing	2
1.3.1 Security Issues	2
1.3.2 Additional Recommendation	2
1.3.3 Note	3
1.4 Security Model	3
<b>Chapter 2 Findings</b>	<b>5</b>
2.1 Security Issue	6
2.1.1 Lack of the access control in the <code>executePath()</code> function	6
2.1.2 Inconsistent handling of the output tokens in the contract <code>MimbokuRouter</code>	9
2.1.3 Potential DoS due to the improper use of ERC20 interfaces	10
2.1.4 Inconsistent access control over funds transfer functions	12
2.1.5 Insufficient checks on a swap path	13
2.1.6 Potential DoS due to the incorrect check in the <code>swapMultiRoutes()</code> function	14
2.1.7 Lack of implementations for modifying the <code>approved</code> list	16
2.1.8 Lock of funds due to the improper check between the variable <code>msg.value</code> and <code>totalNativeValue</code>	17
2.1.9 Potential fee miscalculation in the <code>getPlatformFee()</code> function	18
2.1.10 Potential lock of funds	19
2.1.11 Improper fee deductions in the function <code>_swap()</code>	20
2.2 Recommendation	21
2.2.1 Redundant code	21
2.2.2 Add a length check on the input <code>params</code>	22
2.2.3 Lack of non-zero address checks	24
2.2.4 Unify the logic of returning values	26
2.2.5 Unify the usage of the <code>WIP</code> and <code>WIP9</code> constants	28
2.2.6 Use <code>call()</code> function instead of <code>transfer()</code> function for sending native tokens	28
2.2.7 Add checks when setting the variables <code>platformFee</code> and <code>defaultFee</code>	29
2.2.8 Revise the hardcoded variable <code>WIP9</code>	30
2.3 Note	30
2.3.1 The development of the fee mechanism	30
2.3.2 Fee on transfer tokens are not supported	31
2.3.3 Potential centralization risks	31
2.3.4 Benign router addresses	32

## Report Manifest

Item	Description
Client	Tentou
Target	Mimboku Protocol

## Version History

Version	Date	Description
1.0	May 26, 2025	First release

## Signature

**About BlockSec** BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at [Email](#), [Twitter](#) and [Medium](#).

# Chapter 1 Introduction

## 1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository <sup>1</sup> of Mimboku Protocol of Tentou. The Mimboku Protocol is a DeFi routing system designed to facilitate token swaps across Uniswap V2 and V3-like DEXes. It employs a router-executor architecture to manage swap execution, with support for both ERC20 tokens and the native blockchain currency. Note this audit only focuses on the smart contracts in the following directories/files:

- contracts/

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version ([Version 1](#)), as well as new code (in the following versions) to fix issues in the audit report.

Other files are not within the scope of this audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

Project	Version	Commit Hash
Mimboku Protocol	<a href="#">Version 1</a>	<a href="#">7cad7d861a9c3638fa63162b1788eee4e77d40c3</a>
	<a href="#">Version 2</a>	<a href="#">ec4485e12be3ea73bfae1b897a59beb0372ec3c7</a>

## 1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

---

<sup>1</sup><https://github.com/tentou-tech/aggregator-router-contracts>

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

## 1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

### 1.3.1 Security Issues

- \* Access control
- \* Permission management
- \* Whitelist and blacklist mechanisms
- \* Initialization consistency
- \* Improper use of proxy system
- \* Reentrancy
- \* Denial of Service (DoS)
- \* Untrusted external calls and control flow
- \* Exception handling
- \* Data handling and flow
- \* Events operations
- \* Error-prone randomness
- \* Oracle security
- \* Business logic correctness
- \* Semantic and functional consistency
- \* Emergency mechanisms
- \* Economic and incentive impact

### 1.3.2 Additional Recommendation

- \* Gas efficiency
- \* Code quality and style
- \* Redundant logic and code
- \* Parameter validations
- \* Documentation and comments

### 1.3.3 Note

- \* Centralization risks
- \* Off-chain dependencies
- \* Threat modeling
- \* Protocol-specific assumptions



**Note** The listed checkpoints cover the primary focus areas. Additional checks may be applied depending on the project's design. The audit emphasizes identifying security vulnerabilities rather than verifying standard functionality. When specifications are clear, we assume functional correctness and concentrate on uncovering potential security issues.

## 1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology<sup>2</sup> and Common Weakness Enumeration<sup>3</sup>. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

**Table 1.1:** Vulnerability Severity Classification

Impact	High	High	Medium
	Low	Medium	Low
		High	Low
		Likelihood	

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.

<sup>2</sup>[https://owasp.org/www-community/OWASP\\_Risk\\_Rating\\_Methodology](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology)

<sup>3</sup><https://cwe.mitre.org/>

- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

## Chapter 2 Findings

In total, we found **eleven** potential security issues. Besides, we have **eight** recommendations and **four** notes.

- High Risk: 3
- Medium Risk: 2
- Low Risk: 6
- Recommendation: 8
- Note: 4

ID	Severity	Description	Category	Status
1	High	Lack of the access control in the <code>executePath()</code> function	Security Issue	Fixed
2	High	Inconsistent handling of the output tokens in the contract <code>MimbokuRouter</code>	Security Issue	Fixed
3	High	Potential DoS due to the improper use of ERC20 interfaces	Security Issue	Fixed
4	Medium	Inconsistent access control over funds transfer functions	Security Issue	Fixed
5	Medium	Insufficient checks on a swap path	Security Issue	Fixed
6	Low	Potential DoS due to the incorrect check in the <code>swapMultiroutes()</code> function	Security Issue	Confirmed
7	Low	Lack of implementations for modifying the <code>approved</code> list	Security Issue	Fixed
8	Low	Lock of funds due to the improper check between the variable <code>msg.value</code> and <code>totalNativeValue</code>	Security Issue	Fixed
9	Low	Potential fee miscalculation in the <code>getPlatformFee()</code> function	Security Issue	Fixed
10	Low	Potential lock of funds	Security Issue	Confirmed
11	Low	Improper fee deductions in the function <code>_swap()</code>	Security Issue	Fixed
12	-	Redundant code	Recommendation	Fixed
13	-	Add a length check on the input <code>params</code>	Recommendation	Confirmed
14	-	Lack of non-zero address checks	Recommendation	Partially Fixed
15	-	Unify the logic of returning values	Recommendation	Confirmed
16	-	Unify the usage of the <code>WIP</code> and <code>WIP9</code> constants	Recommendation	Fixed
17	-	Use <code>call()</code> function instead of <code>transfer()</code> function for sending native tokens	Recommendation	Fixed



18	-	Add checks when setting the variables <code>platformFee</code> and <code>defaultFee</code>	Recommendation	Partially Fixed
19	-	Revise the hardcoded variable <code>WIP9</code>	Recommendation	Fixed
20	-	The development of the fee mechanism	Note	-
21	-	Fee on transfer tokens are not supported	Note	-
22	-	Potential centralization risks	Note	-
23	-	Benign router addresses	Note	-

The details are provided in the following sections.

## 2.1 Security Issue

### 2.1.1 Lack of the access control in the `executePath()` function

**Severity** High

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the `MimbokuRouter` contract, users can invoke the `swapMultiroutes()` and `swap()` functions to perform swaps based on provided paths (i.e., the variable `swapRoutes` indicates a swap path). These functions internally invoke the `executePath()` function of the `MimbokuExecutor` contract to execute the swap logic in Uniswap V2 or V3-like pools. However, the `executePath()` function lacks access control, which introduces several potential vulnerabilities:

1. Users can circumvent the fee mechanism by directly invoking the `executePath()` function to perform swaps.
2. Users may suffer from sandwich attacks when directly invoking the `executePath()` function, as the swap executed via the `_executeV2Swap()` and `_executeV3Swap()` functions lack slippage protection.
3. Malicious users can steal ERC20 tokens held by the `MimbokuExecutor` contract. Specifically, the users can invoke the `executePath()` function with an input token set to any ERC20 tokens held by the `MimbokuExecutor` contract.

As a result, both the protocol and its users may suffer fund losses due to the publicly accessible `executePath()` function.

```

20  function executePath(bytes calldata route) external payable returns (uint256 amountOut) {
21      // Decode the route using ExactInputParams
22      Path.ExactInputParams memory params = abi.decode(route, (Path.ExactInputParams));
23
24      // Validate basic parameters
25      require(params.deadline >= block.timestamp, "Deadline expired");
26      require(params.amountIn > 0, "Invalid amount in");
27
28      // Override the tokenIn if it's Native token
29      if (params.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
30          IWIP9(WIP9).deposit{value: msg.value}(); // wrap only what is needed to pay
31          params.swapRoutes[0].tokenIn = WIP9;

```

```
32     }
33
34     uint256 amountIn = params.amountIn;
35     amountOut = 0;
36     for (uint256 i = 0; i < params.swapRoutes.length; i++) {
37         Path.SwapRoute memory routePath = params.swapRoutes[i];
38         if (routePath.poolType == Path.PoolType.V3Pool) {
39             amountOut = _executeV3Swap(amountIn, routePath);
40             amountIn = amountOut;
41         } else {
42             amountOut = _executeV2Swap(amountIn, routePath);
43             amountIn = amountOut;
44         }
45     }
46
47     if (params.recipient == address(0)) {
48         TokenHelper.universalTransfer(
49             params.swapRoutes[params.swapRoutes.length - 1].tokenOut, msg.sender, amountOut
50         );
51     } else {
52         TokenHelper.universalTransfer(
53             params.swapRoutes[params.swapRoutes.length - 1].tokenOut, params.recipient,
54             amountOut
55         );
56     }
57     return amountOut;
58 }
```

**Listing 2.1:** contracts/MimbokuExecutor.sol

```
96 function swapMultiRoutes(Path.ExactInputParams[] memory params)
97     external
98     payable
99     whenNotPaused
100     returns (uint256[] memory amountOuts)
101 {
102     uint256 totalNativeValue = 0;
103     amountOuts = new uint256[](params.length);
104     for (uint256 i = 0; i < params.length; i++) {
105         Path.ExactInputParams memory currParams = params[i];
106
107         require(currParams.swapRoutes.length > 0, "No routes provided");
108         if (currParams.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
109             require(currParams.amountIn != 0, "Amount in must NOT be 0");
110             totalNativeValue += currParams.amountIn;
111         } else {
112             require(msg.value == 0, InvalidNativeValueDepositOnERC20Swap());
113             require(currParams.amountIn > 0, "Amount in must be greater than 0");
114             IERC20(currParams.swapRoutes[0].tokenIn).safeTransferFrom(
115                 msg.sender, address(executor), currParams.amountIn
116             );
117         }
118     }
119 }
```

```
118
119     bool isNative = currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut ==
        TokenHelper.NATIVE_TOKEN;
120     address originalRecipient = currParams.recipient;
121     // if the tokenOut is NATIVE_TOKEN, we need to send the amountOut to the user
122     if (isNative) {
123         // change the recipient to this contract
124         currParams.recipient = address(this);
125
126         // change the tokenOut to WIP
127         currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut = WIP;
128     }
129
130     amountOuts[i] = _swap(currParams);
131
132     // unwrap the WIP to the tokenOut
133     if (isNative) {
134         IWIP(WIP).withdraw(amountOuts[i]);
135         // send the native token to the original recipient
136         payable(originalRecipient).transfer(amountOuts[i]);
137     }
138 }
139
140 require(msg.value >= totalNativeValue, NativeDepositValueMismatch(totalNativeValue, msg.
    value));
141
142 return amountOuts;
143 }
```

**Listing 2.2:** contracts/MimbokuRouter.sol

```
147 function swap(Path.ExactInputParams memory params) external payable whenNotPaused returns (
    uint256 amountOut) {
148     require(params.swapRoutes.length > 0, "No routes provided");
149     if (params.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
150         // Support rebasing tokens by allowing the user to trade the entire balance
151         if (params.amountIn == 0) {
152             params.amountIn = msg.value;
153         } else {
154             require(msg.value == params.amountIn, NativeDepositValueMismatch(params.amountIn,
                msg.value));
155         }
156     } else {
157         require(msg.value == 0, InvalidNativeValueDepositOnERC20Swap());
158         require(params.amountIn > 0, "Amount in must be greater than 0");
159         IERC20(params.swapRoutes[0].tokenIn).safeTransferFrom(msg.sender, address(executor),
            params.amountIn);
160     }
161
162     return _swap(params);
163 }
```

**Listing 2.3:** contracts/MimbokuRouter.sol

**Impact** Both the protocol and its users may suffer fund losses due to the publicly accessible `executePath()` function.

**Suggestion** Add access control to the `executePath()` function (e.g., only allow the contract `MimbokuRouter` to invoke the `executePath()` function).

### 2.1.2 Inconsistent handling of the output tokens in the contract `MimbokuRouter`

**Severity** High

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the `MimbokuRouter` contract, the `swapMultiroutes()` function modifies users' swap paths by replacing the output token (i.e., the variable `tokenOut`) with `WIP` when the specified output token in a path is `NATIVE_TOKEN`. This is necessary because the `MimbokuExecutor` contract is designed to handle swaps only for ERC20 tokens. However, the `swap()` function does not implement similar handling logic. As a result, the `swap()` function reverts when the output token is set to `NATIVE_TOKEN`.

```
147 function swap(Path.ExactInputParams memory params) external payable whenNotPaused returns (
148     uint256 amountOut) {
149     require(params.swapRoutes.length > 0, "No routes provided");
150     if (params.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
151         // Support rebasing tokens by allowing the user to trade the entire balance
152         if (params.amountIn == 0) {
153             params.amountIn = msg.value;
154         } else {
155             require(msg.value == params.amountIn, NativeDepositValueMismatch(params.amountIn,
156                 msg.value));
157         }
158     } else {
159         require(msg.value == 0, InvalidNativeValueDepositOnERC20Swap());
160         require(params.amountIn > 0, "Amount in must be greater than 0");
161         IERC20(params.swapRoutes[0].tokenIn).safeTransferFrom(msg.sender, address(executor),
162             params.amountIn);
163     }
164     return _swap(params);
165 }
```

**Listing 2.4:** contracts/MimbokuRouter.sol

```
96 function swapMultiroutes(Path.ExactInputParams[] memory params)
97     external
98     payable
99     whenNotPaused
100     returns (uint256[] memory amountOuts)
101 {
102     uint256 totalNativeValue = 0;
103     amountOuts = new uint256[](params.length);
104     for (uint256 i = 0; i < params.length; i++) {
105         Path.ExactInputParams memory currParams = params[i];
```

```
106
107     require(currParams.swapRoutes.length > 0, "No routes provided");
108     if (currParams.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
109         require(currParams.amountIn != 0, "Amount in must NOT be 0");
110         totalNativeValue += currParams.amountIn;
111     } else {
112         require(msg.value == 0, InvalidNativeValueDepositOnERC20Swap());
113         require(currParams.amountIn > 0, "Amount in must be greater than 0");
114         IERC20(currParams.swapRoutes[0].tokenIn).safeTransferFrom(
115             msg.sender, address(executor), currParams.amountIn
116         );
117     }
118
119     bool isNative = currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut ==
        TokenHelper.NATIVE_TOKEN;
120     address originalRecipient = currParams.recipient;
121     // if the tokenOut is NATIVE_TOKEN, we need to send the amountOut to the user
122     if (isNative) {
123         // change the recipient to this contract
124         currParams.recipient = address(this);
125
126         // change the tokenOut to WIP
127         currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut = WIP;
128     }
```

**Listing 2.5:** contracts/MimbokuRouter.sol

**Impact** The `swap()` function reverts when the output token is set to `NATIVE_TOKEN`.

**Suggestion** Revise the logic accordingly.

### 2.1.3 Potential DoS due to the improper use of ERC20 interfaces

**Severity** High

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the `MimbokuRouter` contract, the `_swap()` function transfers the output token using OpenZeppelin's `IERC20` interface. Specifically, it relies on the function `transfer()`, which returns a boolean value in the `IERC20` interface. However, this interface is incompatible with certain tokens (e.g., USDT) that do not return a value from their `transfer()` function. As a result, when such tokens are used as output tokens in a swap path, the function `_swap()` function reverts.

```
190     // platform fee
191     uint256 fee = getPlatformFee(amountOut);
192     if (fee > 0) {
193         // transfer the fee to the fee receiver
194         IERC20(lastRoute.tokenOut).transfer(feeReceiver, fee);
195
196         // update the amountOut
197         amountOut -= fee;
```

```
198     }
199
200     // if the originalRecipient is not this contract, we need to send the amountOut to the
        originalRecipient
201     if (originalRecipient != address(this)) {
202         IERC20(lastRoute.tokenOut).transfer(originalRecipient, amountOut);
203
204         // change the recipient back to the originalRecipient
205         params.recipient = originalRecipient;
206     }
```

**Listing 2.6:** contracts/MimbokuRouter.sol

```
9interface IERC20 {
10    /**
11     * @dev Emitted when 'value' tokens are moved from one account ('from') to
12     * another ('to').
13     *
14     * Note that 'value' may be zero.
15     */
16    event Transfer(address indexed from, address indexed to, uint256 value);
17
18    /**
19     * @dev Emitted when the allowance of a 'spender' for an 'owner' is set by
20     * a call to {approve}. 'value' is the new allowance.
21     */
22    event Approval(address indexed owner, address indexed spender, uint256 value);
23
24    /**
25     * @dev Returns the value of tokens in existence.
26     */
27    function totalSupply() external view returns (uint256);
28
29    /**
30     * @dev Returns the value of tokens owned by 'account'.
31     */
32    function balanceOf(address account) external view returns (uint256);
33
34    /**
35     * @dev Moves a 'value' amount of tokens from the caller's account to 'to'.
36     *
37     * Returns a boolean value indicating whether the operation succeeded.
38     *
39     * Emits a {Transfer} event.
40     */
41    function transfer(address to, uint256 value) external returns (bool);
```

**Listing 2.7:** lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol

**Impact** The function `_swap()` function reverts when the output token is incompatible with the interface `IERC20`.

**Suggestion** Unify the use of the `SafeERC20` library for ERC20 token transfers.

### 2.1.4 Inconsistent access control over funds transfer functions

**Severity** Medium

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the [MimbokuRouter](#) contract, the `transferRouterFunds()` function allows approved addresses to transfer assets from the contract. However, the functions `refundIp()` and `sweepTokens()`, which are inherited from the contract [PeripheryPayments](#), are publicly accessible to transfer assets. This inconsistent design allows anyone to steal funds from the [MimbokuRouter](#) contract.

```
223
224  function transferRouterFunds(address[] calldata tokens, uint256[] calldata amounts, address
      dest)
225      external
226      onlyApproved
227  {
228      require(tokens.length == amounts.length, InvalidRouterFundsTransfer());
229      for (uint256 i = 0; i < tokens.length; i++) {
230          if (tokens[i] == TokenHelper.NATIVE_TOKEN) {
231              uint256 amount = amounts[i] == 0 ? tokens[i].universalBalance() : amounts[i];
232              IWIP9(WIP9).deposit{value: amount}();
233              IERC20(WIP9).safeTransfer(dest, amount);
234          } else {
235              IERC20(tokens[i]).safeTransfer(dest, amounts[i] == 0 ? tokens[i].universalBalance()
                  : amounts[i]);
236          }
237      }
```

**Listing 2.8:** contracts/MimbokuRouter.sol

```
19  function unwrapWIP9(uint256 amountMinimum, address recipient) public payable override {
20      uint256 balanceWIP9 = IWIP9(WIP9).balanceOf(address(this));
21      require(balanceWIP9 >= amountMinimum, "Insufficient WIP9");
22
23      if (balanceWIP9 > 0) {
24          IWIP9(WIP9).withdraw(balanceWIP9);
25          TransferHelper.safeTransferIP(recipient, balanceWIP9);
26      }
27  }
```

**Listing 2.9:** contracts/PeripheryPayments.sol

```
30  function sweepToken(address token, uint256 amountMinimum, address recipient) public payable
      override {
31      uint256 balanceToken = IERC20(token).balanceOf(address(this));
32      require(balanceToken >= amountMinimum, "Insufficient token");
33
34      if (balanceToken > 0) {
35          TransferHelper.safeTransfer(token, recipient, balanceToken);
36      }
```

```
37 }
```

### Listing 2.10: contracts/PeripheryPayments.sol

```
40 function refundIP() external payable override {
41     if (address(this).balance > 0) TransferHelper.safeTransferIP(msg.sender, address(this).
        balance);
42 }
```

### Listing 2.11: contracts/PeripheryPayments.sol

**Impact** Anyone can steal funds from the [MimbokuRouter](#) contract.

**Suggestion** Revise the logic accordingly.

## 2.1.5 Insufficient checks on a swap path

**Severity** Medium

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the [MimbokuRouter](#) contract, the `_swap()` function includes a check (Line 172) to ensure that the `tokenIn` and `tokenOut` of the first route (i.e., `swapRoutes[0]`) in a swap path are distinct, aiming to prevent swap failures. However, this check is insufficient and does not validate the consistency of the entire path, leading to three potential vulnerabilities:

1. If an intermediate route has identical `tokenIn` and `tokenOut`, the swap fails.
2. If the swap path includes `NATIVE_TOKEN`, the swap fails.
3. A malicious user can insert an intermediate route where the `tokenIn` does not match the `tokenOut` of the previous route. In this case, the swap uses the tokens held by the contract [MimbokuExecutor](#), potentially allowing the attacker to steal funds from the [MimbokuExecutor](#) contract.

As a result, an invalid swap path may cause the swap to fail or lead to potential fund loss for the protocol.

```
169 function _swap(Path.ExactInputParams memory params) internal returns (uint256 amountOut) {
170     {
171         require(
172             params.swapRoutes[0].tokenIn != params.swapRoutes[0].tokenOut,
173             SameTokenInAndOut(params.swapRoutes[0].tokenIn)
174         );
175
176         // we will change the recipient to this contract for processing platform fee
177         address originalRecipient = params.recipient;
178         params.recipient = address(this);
179
180         Path.SwapRoute memory lastRoute = params.swapRoutes[params.swapRoutes.length - 1];
181
182         uint256 executeValue = params.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN ?
            params.amountIn : 0;
183
184         amountOut = executor.executePath{value: executeValue}(abi.encode(params));
```



```
185
186     if (amountOut < params.amountOutMinimum) {
187         revert SlippageExceeded(amountOut, params.amountOutMinimum);
188     }
189
190     // platform fee
191     uint256 fee = getPlatformFee(amountOut);
192     if (fee > 0) {
193         // transfer the fee to the fee receiver
194         IERC20(lastRoute.tokenOut).transfer(feeReceiver, fee);
195
196         // update the amountOut
197         amountOut -= fee;
198     }
199
200     // if the originalRecipient is not this contract, we need to send the amountOut to the
201     // originalRecipient
202     if (originalRecipient != address(this)) {
203         IERC20(lastRoute.tokenOut).transfer(originalRecipient, amountOut);
204
205         // change the recipient back to the originalRecipient
206         params.recipient = originalRecipient;
207     }
208
209     emit Swap(
210         msg.sender,
211         params.amountIn,
212         params.swapRoutes[0].tokenIn,
213         amountOut,
214         lastRoute.tokenOut,
215         params.recipient
216     );
217 }
```

**Listing 2.12:** contracts/MimbokuRouter.sol

**Impact** An invalid swap path may cause the swap to fail or lead to potential fund loss for the protocol.

**Suggestion** Revise the logic accordingly.

### 2.1.6 Potential DoS due to the incorrect check in the `swapMultiroutes()` function

**Severity** Low

**Status** Confirmed

**Introduced by** [Version 1](#)

**Description** In the `MimbokuRouter` contract, the `swapMultiroutes()` function allows users to perform multiple swaps based on the input `params`. However, the function may revert due to the incorrect check at Line 112. Specifically, if the input includes paths that use different types of input tokens (i.e., both `NATIVE_TOKEN` and ERC20 tokens), the condition (i.e., `msg.value`

`== 0`) fails, resulting in a DoS issue. As a result, the `swapMultiroutes()` function is limited to processing only paths that use the same type of input tokens (i.e., either `NATIVE_TOKEN` or ERC20 tokens).

```
96  function swapMultiroutes(Path.ExactInputParams[] memory params)
97      external
98      payable
99      whenNotPaused
100     returns (uint256[] memory amountOuts)
101 {
102     uint256 totalNativeValue = 0;
103     amountOuts = new uint256[](params.length);
104     for (uint256 i = 0; i < params.length; i++) {
105         Path.ExactInputParams memory currParams = params[i];
106
107         require(currParams.swapRoutes.length > 0, "No routes provided");
108         if (currParams.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
109             require(currParams.amountIn != 0, "Amount in must NOT be 0");
110             totalNativeValue += currParams.amountIn;
111         } else {
112             require(msg.value == 0, InvalidNativeValueDepositOnERC20Swap());
113             require(currParams.amountIn > 0, "Amount in must be greater than 0");
114             IERC20(currParams.swapRoutes[0].tokenIn).safeTransferFrom(
115                 msg.sender, address(executor), currParams.amountIn
116             );
117         }
118
119         bool isNative = currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut ==
            TokenHelper.NATIVE_TOKEN;
120         address originalRecipient = currParams.recipient;
121         // if the tokenOut is NATIVE_TOKEN, we need to send the amountOut to the user
122         if (isNative) {
123             // change the recipient to this contract
124             currParams.recipient = address(this);
125
126             // change the tokenOut to WIP
127             currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut = WIP;
128         }
129
130         amountOuts[i] = _swap(currParams);
131
132         // unwrap the WIP to the tokenOut
133         if (isNative) {
134             IWIP(WIP).withdraw(amountOuts[i]);
135             // send the native token to the original recipient
136             payable(originalRecipient).transfer(amountOuts[i]);
137         }
138     }
139
140     require(msg.value >= totalNativeValue, NativeDepositValueMismatch(totalNativeValue, msg.
        value));
141
142     return amountOuts;
```

```
143 }
```

**Listing 2.13:** contracts/MimbokuRouter.sol

**Impact** The `swapMultiroutes()` function is limited to processing only paths that use the same type of input tokens (i.e., either `NATIVE_TOKEN` or ERC20 tokens).

**Suggestion** Revise the logic accordingly.

**Note** The project stated that this behavior is intentional, and the input tokens provided to the function `swapMultiroutes()` are expected to have the same type.

### 2.1.7 Lack of implementations for modifying the approved list

**Severity** Low

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** The `MimbokuRouter` contract introduces an access control mechanism by inheriting the contract `OnlyApproved`. Specifically, only approved users (i.e., users added to the `approved` list) can invoke the functions (e.g., `transferRouterFunds()`) protected by the modifier `onlyApproved`. However, the contract does not expose any external functions (i.e., to invoke `_addApprovedAddress()` and `_removeApprovedAddress()`) to manage the `approved` list. As a result, the `approved` list becomes immutable and affects the contract's intended functionality.

```
16 function _addApprovedAddress(address _address) internal {
17     approved[_address] = true;
18 }
```

**Listing 2.14:** contracts/OnlyApproved.sol

```
223
224 function transferRouterFunds(address[] calldata tokens, uint256[] calldata amounts, address
    dest)
225     external
226     onlyApproved
227 {
228     require(tokens.length == amounts.length, InvalidRouterFundsTransfer());
229     for (uint256 i = 0; i < tokens.length; i++) {
230         if (tokens[i] == TokenHelper.NATIVE_TOKEN) {
231             uint256 amount = amounts[i] == 0 ? tokens[i].universalBalance() : amounts[i];
232             IWIP(WIP9).deposit{value: amount}();
233             IERC20(WIP9).safeTransfer(dest, amount);
234         } else {
235             IERC20(tokens[i]).safeTransfer(dest, amounts[i] == 0 ? tokens[i].universalBalance()
                : amounts[i]);
236         }
237     }
```

**Listing 2.15:** contracts/MimbokuRouter.sol

```

20  function _removeApprovedAddress(address _address) internal {
21      approved[_address] = false;
22  }

```

**Listing 2.16:** contracts/OnlyApproved.sol

**Impact** The `approved` list is immutable affecting the contract's intended functionality

**Suggestion** Revise the logic accordingly.

### 2.1.8 Lock of funds due to the improper check between the variable `msg.value` and `totalNativeValue`

**Severity** Low

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the `MimbokuRouter` contract, the `swapMultiroutes()` function performs a check at line 140 to ensure that `msg.value` is greater than or equal to the variable `totalNativeValue` (i.e., total used native tokens). However, this check is problematic and may result in funds being locked. Specifically, if a user sends more native tokens than required (i.e., `msg.value > totalNativeValue`), the excess amount remains locked in the `MimbokuRouter` contract.

```

96  function swapMultiroutes(Path.ExactInputParams[] memory params)
97      external
98      payable
99      whenNotPaused
100     returns (uint256[] memory amountOuts)
101  {
102     uint256 totalNativeValue = 0;
103     amountOuts = new uint256[](params.length);
104     for (uint256 i = 0; i < params.length; i++) {
105         Path.ExactInputParams memory currParams = params[i];
106
107         require(currParams.swapRoutes.length > 0, "No routes provided");
108         if (currParams.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
109             require(currParams.amountIn != 0, "Amount in must NOT be 0");
110             totalNativeValue += currParams.amountIn;
111         } else {
112             require(msg.value == 0, InvalidNativeValueDepositOnERC20Swap());
113             require(currParams.amountIn > 0, "Amount in must be greater than 0");
114             IERC20(currParams.swapRoutes[0].tokenIn).safeTransferFrom(
115                 msg.sender, address(executor), currParams.amountIn
116             );
117         }
118
119         bool isNative = currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut ==
120             TokenHelper.NATIVE_TOKEN;
121         address originalRecipient = currParams.recipient;
122         // if the tokenOut is NATIVE_TOKEN, we need to send the amountOut to the user
123         if (isNative) {

```

```
123         // change the recipient to this contract
124         currParams.recipient = address(this);
125
126         // change the tokenOut to WIP
127         currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut = WIP;
128     }
129
130     amountOuts[i] = _swap(currParams);
131
132     // unwrap the WIP to the tokenOut
133     if (isNative) {
134         IWIP(WIP).withdraw(amountOuts[i]);
135         // send the native token to the original recipient
136         payable(originalRecipient).transfer(amountOuts[i]);
137     }
138 }
139
140 require(msg.value >= totalNativeValue, NativeDepositValueMismatch(totalNativeValue, msg.
    value));
141
142 return amountOuts;
143 }
```

**Listing 2.17:** contracts/MimbokuRouter.sol

**Impact** The excess amount of native tokens remains locked in the `MimbokuRouter` contract.

**Suggestion** Revise the check to ensure that the `msg.value` is equal to the `totalNativeValue` variable.

### 2.1.9 Potential fee miscalculation in the `getPlatformFee()` function

**Severity** Low

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the `MimbokuRouter` contract, the annotation (Line 249) of the `getPlatformFee()` function states that a `fee` value of `3000` corresponds to a `0.3%` fee. However, this is incorrect given that the denominator is `10000`. This misleading annotation may result in incorrect fee calculations.

```
241 function getPlatformFee(uint256 amount) internal returns (uint256 feeAmount) {
242     if (amount > 0) {
243         uint256 fee = platformFee;
244         if (isUseCustomFee) {
245             require(customFeeContract != address(0), "Custom fee contract not set");
246             fee = ICustomFee(customFeeContract).feeOf(msg.sender);
247         }
248
249         feeAmount = fee > 0 ? (amount * fee) / 10000 : 0; // 3000 = 0.3%
250     } else {
251         feeAmount = 0;
252     }
```

```
252     }
253 }
```

**Listing 2.18:** contracts/MimbokuRouter.sol

**Impact** This misleading annotation may result in incorrect fee calculations.

**Suggestion** Revise the logic accordingly.

### 2.1.10 Potential lock of funds

**Severity** Low

**Status** Confirmed

**Introduced by** [Version 1](#)

**Description** In the [MimbokuExecutor](#) contract, the [executePath\(\)](#) function performs a series of swaps in either Uniswap V2 or V3 -like pools based on the provided path (i.e., [param.swapRoutes](#)). However, this design can become problematic when a Uniswap V3 -like pool lacks sufficient liquidity. Specifically, if a swap in Uniswap V3 -like pool is attempted under low -liquidity conditions, the input tokens may be only partially consumed, with the remaining tokens left in the contract [MimbokuExecutor](#). As a result, due to the lack of a sweeping or recovery mechanism, the unconsumed tokens remain locked in the contract [MimbokuExecutor](#).

```
20  function executePath(bytes calldata route) external payable returns (uint256 amountOut) {
21      // Decode the route using ExactInputParams
22      Path.ExactInputParams memory params = abi.decode(route, (Path.ExactInputParams));
23
24      // Validate basic parameters
25      require(params.deadline >= block.timestamp, "Deadline expired");
26      require(params.amountIn > 0, "Invalid amount in");
27
28      // Override the tokenIn if it's Native token
29      if (params.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
30          IWIP9(WIP9).deposit{value: msg.value}(); // wrap only what is needed to pay
31          params.swapRoutes[0].tokenIn = WIP9;
32      }
33
34      uint256 amountIn = params.amountIn;
35      amountOut = 0;
36      for (uint256 i = 0; i < params.swapRoutes.length; i++) {
37          Path.SwapRoute memory routePath = params.swapRoutes[i];
38          if (routePath.poolType == Path.PoolType.V3Pool) {
39              amountOut = _executeV3Swap(amountIn, routePath);
40              amountIn = amountOut;
41          } else {
42              amountOut = _executeV2Swap(amountIn, routePath);
43              amountIn = amountOut;
44          }
45      }
46
47      if (params.recipient == address(0)) {
48          TokenHelper.universalTransfer(
```

```
49         params.swapRoutes[params.swapRoutes.length - 1].tokenOut, msg.sender, amountOut
50     );
51 } else {
52     TokenHelper.universalTransfer(
53         params.swapRoutes[params.swapRoutes.length - 1].tokenOut, params.recipient,
54         amountOut
55     );
56 }
57 return amountOut;
58 }
```

**Listing 2.19:** contracts/MimbokuExecutor.sol

**Impact** The unconsumed tokens remain locked in the contract [MimbokuExecutor](#).

**Suggestion** Revise the logic accordingly.

**Note** The project decided to keep this design and stated that users are responsible for the provided input data. Specifically, slippage should be carefully set to prevent funds from being locked in the contract [MimbokuExecutor](#).

### 2.1.11 Improper fee deductions in the function `_swap()`

**Severity** Low

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the [MimbokuRouter](#) contract, the `_swap()` function performs the slippage check (Line 186) before the fee deduction (Line 194). As a result, the final amount received by users may be less than the expected amount (i.e., `amountOutMinimum`) after the fee deduction.

```
169 function _swap(Path.ExactInputParams memory params) internal returns (uint256 amountOut) {
170     {
171         require(
172             params.swapRoutes[0].tokenIn != params.swapRoutes[0].tokenOut,
173             SameTokenInAndOut(params.swapRoutes[0].tokenIn)
174         );
175
176         // we will change the recipient to this contract for processing platform fee
177         address originalRecipient = params.recipient;
178         params.recipient = address(this);
179
180         Path.SwapRoute memory lastRoute = params.swapRoutes[params.swapRoutes.length - 1];
181
182         uint256 executeValue = params.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN ?
183             params.amountIn : 0;
184
185         amountOut = executor.executePath{value: executeValue}(abi.encode(params));
186
187         if (amountOut < params.amountOutMinimum) {
188             revert SlippageExceeded(amountOut, params.amountOutMinimum);
189         }
190     }
191 }
```

```
189
190     // platform fee
191     uint256 fee = getPlatformFee(amountOut);
192     if (fee > 0) {
193         // transfer the fee to the fee receiver
194         IERC20(lastRoute.tokenOut).transfer(feeReceiver, fee);
195
196         // update the amountOut
197         amountOut -= fee;
198     }
199
200     // if the originalRecipient is not this contract, we need to send the amountOut to the
201     // originalRecipient
202     if (originalRecipient != address(this)) {
203         IERC20(lastRoute.tokenOut).transfer(originalRecipient, amountOut);
204
205         // change the recipient back to the originalRecipient
206         params.recipient = originalRecipient;
207     }
208
209     emit Swap(
210         msg.sender,
211         params.amountIn,
212         params.swapRoutes[0].tokenIn,
213         amountOut,
214         lastRoute.tokenOut,
215         params.recipient
216     );
217 }
```

**Listing 2.20:** contracts/MimbokuRouter.sol

**Impact** The final amount received by users may be less than the expected amount.

**Suggestion** Deduct the fee before the slippage check.

## 2.2 Recommendation

### 2.2.1 Redundant code

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** There are several unused interfaces, variables, functions, errors. It is recommended to remove them for better code readability. Specifically, the following code should be removed or revised.

1. Unused abicoder.

```
3pragma abicoder v2;
```

**Listing 2.21:** contracts/Multicall.sol



## 2. Unused interfaces.

```
4import {ISignatureTransfer} from "contracts/interfaces/ISignatureTransfer.sol";
```

**Listing 2.22:** contracts/MimbokuRouter.sol

## 3. Unused variables.

```
38 uint256 public constant REFERRAL_WITH_FEE_THRESHOLD = 1 << 31;
39 uint256 public constant FEE_DENOM = 1e18;
```

**Listing 2.23:** contracts/MimbokuRouter.sol

## 4. Unused functions.

```
48 function pay(address token, address payer, address recipient, uint256 value) internal {
49     if (token == WIP9 && address(this).balance >= value) {
50         // pay with WIP9
51         IWIP9(WIP9).deposit{value: value}(); // wrap only what is needed to pay
52         IWIP9(WIP9).transfer(recipient, value);
53     } else if (payer == address(this)) {
54         // pay with tokens already in the contract (for the exact input multihop case)
55         TransferHelper.safeTransfer(token, recipient, value);
56     } else {
57         // pull payment
58         TransferHelper.safeTransferFrom(token, payer, recipient, value);
59     }
60 }
```

**Listing 2.24:** contracts/PeripheryPayments.sol

## 5. Unused errors.

```
16 error InsufficientWIP9Balance(uint256 contract_balance, uint256 amount_out);
```

**Listing 2.25:** contracts/MimbokuExecutor.sol

## 6. Redundant code logic.

```
14 uint256 public defaultFee = 1000;
```

**Listing 2.26:** contracts/fee/CustomFee.sol

## 7. The contract `Multicall` is redundant.

**Suggestion** Remove the redundant code.

### 2.2.2 Add a length check on the input `params`

**Status** Confirmed

**Introduced by** Version 1

**Description** In the `MimbokuRouter` contract, the `swapMultiroutes()` function supports multiple `tokenIn` and `tokenOut` exchange paths. However, it does not check the length of the input `params`. It is recommended to add a length check (i.e., ensure that the length of the input `params` is greater than one) on the input `params` for gas optimization.

```
96  function swapMultiRoutes(Path.ExactInputParams[] memory params)
97      external
98      payable
99      whenNotPaused
100     returns (uint256[] memory amountOuts)
101 {
102     uint256 totalNativeValue = 0;
103     amountOuts = new uint256[](params.length);
104     for (uint256 i = 0; i < params.length; i++) {
105         Path.ExactInputParams memory currParams = params[i];
106
107         require(currParams.swapRoutes.length > 0, "No routes provided");
108         if (currParams.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
109             require(currParams.amountIn != 0, "Amount in must NOT be 0");
110             totalNativeValue += currParams.amountIn;
111         } else {
112             require(msg.value == 0, InvalidNativeValueDepositOnERC20Swap());
113             require(currParams.amountIn > 0, "Amount in must be greater than 0");
114             IERC20(currParams.swapRoutes[0].tokenIn).safeTransferFrom(
115                 msg.sender, address(executor), currParams.amountIn
116             );
117         }
118
119         bool isNative = currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut ==
            TokenHelper.NATIVE_TOKEN;
120         address originalRecipient = currParams.recipient;
121         // if the tokenOut is NATIVE_TOKEN, we need to send the amountOut to the user
122         if (isNative) {
123             // change the recipient to this contract
124             currParams.recipient = address(this);
125
126             // change the tokenOut to WIP
127             currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut = WIP;
128         }
129
130         amountOuts[i] = _swap(currParams);
131
132         // unwrap the WIP to the tokenOut
133         if (isNative) {
134             IWIP(WIP).withdraw(amountOuts[i]);
135             // send the native token to the original recipient
136             payable(originalRecipient).transfer(amountOuts[i]);
137         }
138     }
139
140     require(msg.value >= totalNativeValue, NativeDepositValueMismatch(totalNativeValue, msg.
        value));
141
142     return amountOuts;
143 }
```

**Listing 2.27:** contracts/MimbokuRouter.sol

**Suggestion** Add a length check on the input `params`.

### 2.2.3 Lack of non-zero address checks

**Status** Partially Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the `MimbokuRouter` contract, the functions `constructor()`, `setFeeReceiver()`, and `setCustomFeeContract()` do not check that certain addresses (e.g., `_feeReceiver`) are non-zero. Additionally, the `swapMultiroutes()` and `swap()` functions do not check whether the variable `params.recipient` is the zero address. It is recommended to add these checks to prevent potential mis-operations.

```
48  constructor(address _owner, address _wip, address _executor, address _feeReceiver)
49      Ownable(_owner)
50      PeripheryImmutableState(_wip)
51  {
52      WIP = _wip;
53      _addApprovedAddress(_owner);
54      executor = IMimbokuExecutor(_executor);
55
56      feeReceiver = _feeReceiver;
57
58      platformFee = 0;
59
60      // by default, we don't use custom fee contract
61      isUseCustomFee = false;
62  }
```

**Listing 2.28:** contracts/MimbokuRouter.sol

```
74  /// @dev Set the fee receiver
75  function setFeeReceiver(address _feeReceiver) external onlyOwner {
76      feeReceiver = _feeReceiver;
77  }
```

**Listing 2.29:** contracts/MimbokuRouter.sol

```
85  function setCustomFeeContract(address _customFeeContract) external onlyOwner {
86      customFeeContract = _customFeeContract;
87  }
```

**Listing 2.30:** contracts/MimbokuRouter.sol

```
96  function swapMultiroutes(Path.ExactInputParams[] memory params)
97      external
98      payable
99      whenNotPaused
100     returns (uint256[] memory amountOuts)
101  {
102     uint256 totalNativeValue = 0;
103     amountOuts = new uint256[](params.length);
104     for (uint256 i = 0; i < params.length; i++) {
```

```
105     Path.ExactInputParams memory currParams = params[i];
106
107     require(currParams.swapRoutes.length > 0, "No routes provided");
108     if (currParams.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
109         require(currParams.amountIn != 0, "Amount in must NOT be 0");
110         totalNativeValue += currParams.amountIn;
111     } else {
112         require(msg.value == 0, InvalidNativeValueDepositOnERC20Swap());
113         require(currParams.amountIn > 0, "Amount in must be greater than 0");
114         IERC20(currParams.swapRoutes[0].tokenIn).safeTransferFrom(
115             msg.sender, address(executor), currParams.amountIn
116         );
117     }
118
119     bool isNative = currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut ==
        TokenHelper.NATIVE_TOKEN;
120     address originalRecipient = currParams.recipient;
121     // if the tokenOut is NATIVE_TOKEN, we need to send the amountOut to the user
122     if (isNative) {
123         // change the recipient to this contract
124         currParams.recipient = address(this);
125
126         // change the tokenOut to WIP
127         currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut = WIP;
128     }
129
130     amountOuts[i] = _swap(currParams);
131
132     // unwrap the WIP to the tokenOut
133     if (isNative) {
134         IWIP(WIP).withdraw(amountOuts[i]);
135         // send the native token to the original recipient
136         payable(originalRecipient).transfer(amountOuts[i]);
137     }
138 }
139
140 require(msg.value >= totalNativeValue, NativeDepositValueMismatch(totalNativeValue, msg.
    value));
141
142 return amountOuts;
143 }
```

**Listing 2.31:** contracts/MimbokuRouter.sol

```
146 /// @param params All information about the tokens being swapped
147 function swap(Path.ExactInputParams memory params) external payable whenNotPaused returns (
    uint256 amountOut) {
148     require(params.swapRoutes.length > 0, "No routes provided");
149     if (params.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
150         // Support rebasing tokens by allowing the user to trade the entire balance
151         if (params.amountIn == 0) {
152             params.amountIn = msg.value;
153         } else {
```

```
154         require(msg.value == params.amountIn, NativeDepositValueMismatch(params.amountIn,
155             msg.value));
156     }
157 } else {
158     require(msg.value == 0, InvalidNativeValueDepositOnERC20Swap());
159     require(params.amountIn > 0, "Amount in must be greater than 0");
160     IERC20(params.swapRoutes[0].tokenIn).safeTransferFrom(msg.sender, address(executor),
161         params.amountIn);
162 }
163
164 return _swap(params);
165 }
```

**Listing 2.32:** contracts/MimbokuRouter.sol

**Suggestion** Add non-zero address checks accordingly.

**Note** The project only added non-zero address checks for the `recipient` of the input `params` in the function `swapMultiroutes()`.

## 2.2.4 Unify the logic of returning values

**Status** Confirmed

**Introduced by** Version 1

**Description** In the `MimbokuRouter` and `MimbokuExecutor` contracts, there is an inconsistency in how return values are handled. Specifically, some functions use implicit logic to return (i.e., the `_swap()` and `getPlatformFee()` functions), while others use explicit return statements (i.e., the `swap()`, `swapMultiroutes()`, `swap()`, `executePath()`, `_executeV3Swap()`, and `_executeV2Swap()` functions). It is recommended to unify the return logic across the contracts to improve code readability and maintainability. Note that not all the code of the functions mentioned above are listed.

```
147 function swap(Path.ExactInputParams memory params) external payable whenNotPaused returns (
148     uint256 amountOut) {
149     require(params.swapRoutes.length > 0, "No routes provided");
150     if (params.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN) {
151         // Support rebasing tokens by allowing the user to trade the entire balance
152         if (params.amountIn == 0) {
153             params.amountIn = msg.value;
154         } else {
155             require(msg.value == params.amountIn, NativeDepositValueMismatch(params.amountIn,
156                 msg.value));
157         }
158     } else {
159         require(msg.value == 0, InvalidNativeValueDepositOnERC20Swap());
160         require(params.amountIn > 0, "Amount in must be greater than 0");
161         IERC20(params.swapRoutes[0].tokenIn).safeTransferFrom(msg.sender, address(executor),
162             params.amountIn);
163     }
164
165     return _swap(params);
166 }
```

```
163 }
```

### Listing 2.33: contracts/MimbokuRouter.sol

```
169 function _swap(Path.ExactInputParams memory params) internal returns (uint256 amountOut) {
170     {
171         require(
172             params.swapRoutes[0].tokenIn != params.swapRoutes[0].tokenOut,
173             SameTokenInAndOut(params.swapRoutes[0].tokenIn)
174         );
175
176         // we will change the recipient to this contract for processing platform fee
177         address originalRecipient = params.recipient;
178         params.recipient = address(this);
179
180         Path.SwapRoute memory lastRoute = params.swapRoutes[params.swapRoutes.length - 1];
181
182         uint256 executeValue = params.swapRoutes[0].tokenIn == TokenHelper.NATIVE_TOKEN ?
            params.amountIn : 0;
183
184         amountOut = executor.executePath{value: executeValue}(abi.encode(params));
185
186         if (amountOut < params.amountOutMinimum) {
187             revert SlippageExceeded(amountOut, params.amountOutMinimum);
188         }
189
190         // platform fee
191         uint256 fee = getPlatformFee(amountOut);
192         if (fee > 0) {
193             // transfer the fee to the fee receiver
194             IERC20(lastRoute.tokenOut).transfer(feeReceiver, fee);
195
196             // update the amountOut
197             amountOut -= fee;
198         }
199
200         // if the originalRecipient is not this contract, we need to send the amountOut to the
            originalRecipient
201         if (originalRecipient != address(this)) {
202             IERC20(lastRoute.tokenOut).transfer(originalRecipient, amountOut);
203
204             // change the recipient back to the originalRecipient
205             params.recipient = originalRecipient;
206         }
207
208         emit Swap(
209             msg.sender,
210             params.amountIn,
211             params.swapRoutes[0].tokenIn,
212             amountOut,
213             lastRoute.tokenOut,
214             params.recipient
215         );
216     }
217 }
```

```
216     }
217 }
```

**Listing 2.34:** contracts/MimbokuRouter.sol

**Suggestion** Unify the logic of returning values.

## 2.2.5 Unify the usage of the WIP and WIP9 constants

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** The constants `WIP9` in the contract `PeripheryImmutableState` and `WIP` in the contract `MimbokuRouter` are both assigned the same value (i.e., `_wip`). It is recommended to unify the usage of the `WIP` and `WIP9` constants.

```
7  address public immutable WIP9;
```

**Listing 2.35:** contracts/PeripheryImmutableState.sol

```
40  address public immutable WIP;
```

**Listing 2.36:** contracts/MimbokuRouter.sol

```
48  constructor(address _owner, address _wip, address _executor, address _feeReceiver)
49      Ownable(_owner)
50      PeripheryImmutableState(_wip)
51  {
52      WIP = _wip;
53      _addApprovedAddress(_owner);
54      executor = IMimbokuExecutor(_executor);
55
56      feeReceiver = _feeReceiver;
57
58      platformFee = 0;
59
60      // by default, we don't use custom fee contract
61      isUseCustomFee = false;
62  }
```

**Listing 2.37:** contracts/MimbokuRouter.sol

**Suggestion** Unify the usage of the `WIP` and `WIP9` constants.

## 2.2.6 Use `call()` function instead of `transfer()` function for sending native tokens

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the `MimbokuRouter` contract, the `swapMultiroutes()` function uses the low-level `transfer()` function to send the native tokens. It is recommended to use the low-level `call()` function instead.

```
119         bool isNative = currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut ==
            TokenHelper.NATIVE_TOKEN;
120         address originalRecipient = currParams.recipient;
121         // if the tokenOut is NATIVE_TOKEN, we need to send the amountOut to the user
122         if (isNative) {
123             // change the recipient to this contract
124             currParams.recipient = address(this);
125
126             // change the tokenOut to WIP
127             currParams.swapRoutes[currParams.swapRoutes.length - 1].tokenOut = WIP;
128         }
129
130         amountOuts[i] = _swap(currParams);
131
132         // unwrap the WIP to the tokenOut
133         if (isNative) {
134             IWIP(WIP).withdraw(amountOuts[i]);
135             // send the native token to the original recipient
136             payable(originalRecipient).transfer(amountOuts[i]);
137         }
138     }
139
140     require(msg.value >= totalNativeValue, NativeDepositValueMismatch(totalNativeValue, msg.
        value));
141
142     return amountOuts;
143 }
```

**Listing 2.38:** contracts/MimbokuRouter.sol

**Suggestion** Use `call()` function instead of `transfer()` function for sending native tokens

## 2.2.7 Add checks when setting the variables `platformFee` and `defaultFee`

**Status** Partially Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** The `getPlatformFee()` function of the `MimbokuRouter` contract calculates the fee based on the variables `platformFee` (in the `MimbokuRouter` contract) or `defaultFee` (in the contract `CustomFee`). It is recommended to add checks (e.g., upper bound checks) when setting the variables `platformFee` and `defaultFee`.

```
16     constructor(address _owner, address _tokenContract, uint256 _defaultFee) Ownable(_owner) {
17         tokenContract = _tokenContract;
18         defaultFee = _defaultFee;
19     }
```

**Listing 2.39:** contracts/fee/CustomFee.sol

```
36     function setDefaultFee(uint256 _defaultFee) public onlyOwner {
37         defaultFee = _defaultFee;
38     }
```



#### Listing 2.40: contracts/fee/CustomFee.sol

```
80  function setPlatformFee(uint256 _platformFee) external onlyOwner {
81      platformFee = _platformFee;
82  }
```

#### Listing 2.41: contracts/MimbokuRouter.sol

```
241  function getPlatformFee(uint256 amount) internal returns (uint256 feeAmount) {
242      if (amount > 0) {
243          uint256 fee = platformFee;
244          if (isUseCustomFee) {
245              require(customFeeContract != address(0), "Custom fee contract not set");
246              fee = ICustomFee(customFeeContract).feeOf(msg.sender);
247          }
248
249          feeAmount = fee > 0 ? (amount * fee) / 10000 : 0; // 3000 = 0.3%
250      } else {
251          feeAmount = 0;
252      }
253  }
```

#### Listing 2.42: contracts/MimbokuRouter.sol

**Suggestion** Add checks when setting the variables `platformFee` and `defaultFee`.

**Note** The project only added checks on the variable `platformFee`. The contract `CustomFee` is still under development.

### 2.2.8 Revise the hardcoded variable WIP9

**Status** Fixed in [Version 2](#)

**Introduced by** [Version 1](#)

**Description** In the `MimbokuExecutor` contract, the variable `WIP9` is hardcoded as the `WIP` token address specific to the Story chain. This design choice effectively limits the contract's deployment to the Story chain only. It is recommended to revise the value assignment for the variable `WIP9`.

```
18  address public constant WIP9 = address(0x1514000000000000000000000000000000000000); // WIP9
    address
```

#### Listing 2.43: contracts/MimbokuExecutor.sol

**Suggestion** Revise the hardcoded `WIP9` address.

## 2.3 Note

### 2.3.1 The development of the fee mechanism

**Introduced by** [Version 1](#)

**Description** In the `CustomFee` contract, the `feeOf()` function calculates the fee based on users' balance of the token `tokenContract`. Specifically, users do not pay the fee when they hold the token `tokenContract`. However, this design allows bad actors to circumvent the fee mechanism by transferring the token `tokenContract` in between multiple accounts. Additionally, in the `MimbokuRouter` contract, the function `getPlatformFee()` contains a rounding down issue when calculating the fee amount. Specifically, users pay zero fee when performing a swap with a small `amount`. The project states that the fee mechanism is still under development. It is important to implement a proper fee mechanism to mitigate the potential issues mentioned above.

```
31 function feeOf(address user) public view override whenNotPaused returns (uint256) {
32     uint256 balance = IERC721(tokenContract).balanceOf(user);
33     return balance > 0 ? 0 : defaultFee;
34 }
```

**Listing 2.44:** contracts/fee/CustomFee.sol

```
241 function getPlatformFee(uint256 amount) internal returns (uint256 feeAmount) {
242     if (amount > 0) {
243         uint256 fee = platformFee;
244         if (isUseCustomFee) {
245             require(customFeeContract != address(0), "Custom fee contract not set");
246             fee = ICustomFee(customFeeContract).feeOf(msg.sender);
247         }
248
249         feeAmount = fee > 0 ? (amount * fee) / 10000 : 0; // 3000 = 0.3%
250     } else {
251         feeAmount = 0;
252     }
253 }
```

**Listing 2.45:** contracts/MimbokuRouter.sol

### 2.3.2 Fee on transfer tokens are not supported

**Introduced by** [Version 1](#)

**Description** The protocol facilitates token exchanges through the `MimbokuRouter` contract. It is important to note that the current implementation does not support tokens with transfer fees (i.e., fee-on-transfer tokens).

### 2.3.3 Potential centralization risks

**Introduced by** [Version 1](#)

**Description** In the protocol, several privileged roles (e.g., the `owner` role) can conduct sensitive operations, which introduces potential centralization risks. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

### 2.3.4 Benign router addresses

**Introduced by** [Version 1](#)

**Description** In the [MimbokuRouter](#) contract, users can specify arbitrary router addresses (in the input [params](#)) when invoking the [swapMultiroutes\(\)](#) and [swap\(\)](#) functions. It is important to ensure that the specified router addresses are benign.

