Security Audit Report for swap-router-v1

Date: August 22, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	DeBankDeFi
Target	swap-router-v1

Version History

Version	Date	Description
1.0	August 22, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)
SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C
Date: 2025-08-26 03:51:04+0000

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 [1] of swap-router-v1 of DeBankDeFi.

The protocol is a decentralized exchange (DEX) router that facilitates efficient token swaps by integrating multiple aggregators through a flexible adapter pattern, while ensuring robust fee management. Its core component, the DEX Aggregator, is a smart contract system that optimizes trades by splitting orders across various DEXs to secure better prices and minimize slippage for users.

Note this audit only focuses on the smart contracts in the following directories/files:

- src/*

Other files are not within the scope of the 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.

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.

Project	Version	Commit Hash
swap-router-v1	`Version 1`	`c776ccc3efd4b601e9bebea62d309fe2529683e3`
	`Version 2`	`73958699a49d4c43096874f006abf0ee64f49014`

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

[1] `https://github.com/DeBankDeFi/swap-router-v1`

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.

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 the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency
* Emergency mechanism
* Economic and incentive impact

1.3.2 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

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 [2] and Common Weakness Enumeration [3]. 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 \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

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

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

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

- Medium Risk: 3
- Low Risk: 2
- Recommendation: 8
- Note: 5

ID	Severity	Description	Category	Status
1	Medium	Improper `pause()` mechanism causes the contract to be permanently paused	Security Issue	Confirmed
2	Medium	Fee mechanism can be circumvented	Security Issue	Confirmed
3	Medium	Pause and unpause conflict between different admins	Security Issue	Fixed
4	Low	Fee deduction after slippage check may reduce actual user received amount	Security Issue	Fixed
5	Low	Lack of refund mechanism in the contract `Executor`	Security Issue	Fixed
6	-	Inconsistent check	Recommendation	Confirmed
7	-	Non zero address checks	Recommendation	Confirmed
8	-	Ensure proper checks on parameter `_admins`	Recommendation	Confirmed
9	-	Check the existence of the adapter before adding it	Recommendation	Fixed
10	-	Spelling and comment errors	Recommendation	Fixed
11	-	Redundant code	Recommendation	Fixed
12	-	Add allowance revocation logic after swap operation	Recommendation	Fixed
13	-	Replace the function `transfer()` with the function `call()`	Recommendation	Fixed
14	-	Security consideration for delegatecall-based executor architecture	Note	-
15	-	Security audit assumptions on trusted executors	Note	-
16	-	Potential centralization risks	Note	-
17	-	Weird ERC20 tokens	Note	-
18	-	External pool/router without refund mechanism may cause unnecessary slippage	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Improper `pause()` mechanism causes the contract to be permanently paused

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description The function `unpause()` cannot be successfully executed when the value of the `pauseCount` variable is greater than or equal to 2. As a result, if two or more admins invoke the `pause()` function, the contract will become permanently paused, as it can no longer be unpaused.

```
function pause() external onlyAdmin {
    if (!adminPaused[msg.sender]) {
        adminPaused[msg.sender] = true;
        pauseCount++;
        if (pauseCount > 0 && !paused) {
            paused = true;
        }
        emit Paused(msg.sender);
    }
}

function unpause() external onlyAdmin {
    require(pauseCount < 2, "Admin: cannot unpause when multiple admins paused");
    require(pauseCount > 0, "Admin: not paused");
    if (adminPaused[msg.sender]) {
        adminPaused[msg.sender] = false;
    }
    pauseCount--;
    if (pauseCount == 0) {
        paused = false;
    }

    emit Unpaused(msg.sender);
}
```

Listing 2.1: src/library/Admin.sol

Impact The contract is permanently paused.

Suggestion Revise the logic accordingly.

Feedback from the project This is by design. If the two admins pause the contract, the paused state cannot be released, and the contract will be paused permanently.

2.1.2 Fee mechanism can be circumvented

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description In both the contracts `DexSwap` and `Router`, the function `swap()` parameters (such as `feeReceiver` and `feeRate`) are entirely user-controlled. Users can set `feeRate` to be zero or assign feeReceiver to be their own addresses, allowing them to circumvent protocol fees. This could result in the protocol failing to collect its intended fees.

```
function swap(SwapParams memory params) external payable whenNotPaused nonReentrant {
    _swap(params);
}
```

Listing 2.2: src/aggregatorRouter/DexSwap.sol

```
struct SwapParams {
    string aggregatorId;
    address fromToken;
    uint256 fromTokenAmount;
    address toToken;
    uint256 minAmountOut;
    uint256 feeRate;
    bool feeOnFromToken;
    address feeReceiver;
    bytes data;
}
```

Listing 2.3: src/aggregatorRouter/DexSwap.sol

```
function swap(
    address fromToken,
    uint256 fromTokenAmount,
    address toToken,
    uint256 minAmountOut,
    bool feeOnFromToken,
    uint256 feeRate,
    address feeReceiver,
    Utils.MultiPath[] calldata paths
) external payable whenNotPaused nonReentrant {
    require(fromToken != toToken, "Router: fromToken and toToken cannot be the same");
    require(feeRate <= maxFeeRate, "Router: Fee Rate is too big");
    require(msg.value == (fromToken == UniversalERC20.ETH ? fromTokenAmount : 0), "Router:
        Incorrect msg.value");
    uint256 feeAmount;

    // charge fee
    if (feeOnFromToken) {
        (fromTokenAmount, feeAmount) = chargeFee(fromToken, feeOnFromToken, fromTokenAmount,
            feeRate, feeReceiver);
    }
    // deposit to executor
    if (fromToken != UniversalERC20.ETH) {
        IERC20(fromToken).safeTransferFrom(msg.sender, address(executor), fromTokenAmount);
    }

    uint256 balanceBefore = IERC20(toToken).universalBalanceOf(address(this));

```

```
    //execute swap, transfer token to executor
    executor.executeMegaSwap{value: fromToken == UniversalERC20.ETH ? fromTokenAmount : 0}(
        IERC20(fromToken), IERC20(toToken), paths
    );

    uint256 receivedAmount = IERC20(toToken).universalBalanceOf(address(this)) - balanceBefore;

    // charge fee
    if (!feeOnFromToken) {
        (receivedAmount, feeAmount) = chargeFee(toToken, feeOnFromToken, receivedAmount,
            feeRate, feeReceiver);
    }

    // check slippage
    require(receivedAmount >= minAmountOut, "Router: Slippage Limit Exceeded");

    // transfer out
    IERC20(toToken).universalTransfer(payable(msg.sender), receivedAmount);
    emit Swap(msg.sender, address(fromToken), fromTokenAmount, address(toToken), receivedAmount
        , feeAmount);
}
```

Listing 2.4: src/router/Router.sol

```
function _swap(SwapParams memory params) internal {
    Adapter storage adapter = adapters[params.aggregatorId];

    // 1. check params
    _validateSwapParams(params, adapter);

    uint256 feeAmount;
    uint256 receivedAmount;

    // 2. charge fee on fromToken if needed
    if (params.feeOnFromToken) {
        (params.fromTokenAmount, feeAmount) = _chargeFee(
            params.fromToken, params.feeOnFromToken, params.fromTokenAmount, params.feeRate,
                params.feeReceiver
        );
    }

    // 3. transfer fromToken
    if (params.fromToken != UniversalERC20.ETH) {
        IERC20(params.fromToken).safeTransferFrom(msg.sender, address(spender), params.
            fromTokenAmount);
    }

    // 4. execute swap
    {
        uint256 balanceBefore = IERC20(params.toToken).universalBalanceOf(address(this));
        spender.swap{value: params.fromToken == UniversalERC20.ETH ? params.fromTokenAmount :
            0}(
            adapter.addr,
```

```
                abi.encodeWithSelector(
                    adapter.selector, params.fromToken, params.toToken, msg.sender, params.data
                )
            );
            receivedAmount = IERC20(params.toToken).universalBalanceOf(address(this)) -
                balanceBefore;
        }

        // 5. check slippage
        if (receivedAmount < params.minAmountOut) revert RouterError.SlippageLimitExceeded();

        // 6. charge fee on toToken if needed
        if (!params.feeOnFromToken) {
            (receivedAmount, feeAmount) =
                _chargeFee(params.toToken, params.feeOnFromToken, receivedAmount, params.feeRate,
                    params.feeReceiver);
        }
```

Listing 2.5: src/aggregatorRouter/DexSwap.sol

Impact This could result in the protocol failing to collect its intended fees.

Suggestion Revise the logic accordingly.

Feedback from the project This is by design.

2.1.3 Pause and unpause conflict between different admins

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `unpause()` allows `admin 1` to unpause a pause state initiated by `admin 2`. This creates an inconsistent workflow:

1. `admin 1` invokes the function `pause()`, setting `adminPaused[admin 1] = true`;
2. `admin 2` invokes the function `unpause()`;
3. When `admin 1` attempts to invoke the function `pause()` again, the `!adminPaused[admin 1]` check fails, preventing `admin 1` from pausing again.

```
function pause() external onlyAdmin {
    if (!adminPaused[msg.sender]) {
        adminPaused[msg.sender] = true;
        pauseCount++;
        if (pauseCount > 0 && !paused) {
            paused = true;
        }
        emit Paused(msg.sender);
    }
}

function unpause() external onlyAdmin {
    require(pauseCount < 2, "Admin: cannot unpause when multiple admins paused");
    require(pauseCount > 0, "Admin: not paused");
```

```
        if (adminPaused[msg.sender]) {
            adminPaused[msg.sender] = false;
        }
        pauseCount--;
        if (pauseCount == 0) {
            paused = false;
        }

        emit Unpaused(msg.sender);
    }
```

Listing 2.6: src/library/Admin.sol

Impact Admins lose the ability to pause the system after being unilaterally unpaused by another admin.

Suggestion Revise the logic accordingly.

2.1.4 Fee deduction after slippage check may reduce actual user received amount

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description When fees are deducted from `toToken`, the function `swap()` in the contract `DexSwap` first checks whether `receivedAmount` meets the slippage requirement (`params.minAmountOut`). Only after this check does it deduct the fee and send the remaining `toToken` to the user. This creates a scenario where the user may receive less `toToken` than the `receivedAmount` they expected.

```
        // 5. check slippage
        if (receivedAmount < params.minAmountOut) revert RouterError.SlippageLimitExceeded();

        // 6. charge fee on toToken if needed
        if (!params.feeOnFromToken) {
            (receivedAmount, feeAmount) =
                _chargeFee(params.toToken, params.feeOnFromToken, receivedAmount, params.feeRate,
                    params.feeReceiver);
        }
```

Listing 2.7: src/aggregatorRouter/DexSwap.sol

Impact This creates a scenario where the user may receive less `toToken` than they expected.

Suggestion First execute the function `_chargeFee()`, then verify if `receivedAmount` meets the slippage requirement.

2.1.5 Lack of refund mechanism in the contract `Executor`

Severity Low

Status Fixed in `Version 2`

Introduced by Version 1

Description The function `executeMultiPath()` in the contract `Executor` facilitates multi-path token swaps via whitelisted adapters. However, this design contains a critical flaw, that it lacks a refund mechanism for residual tokens when swaps cannot be fully executed (e.g., due to insufficient liquidity). If a swap partially succeeds, any unused `fromToken` will remain permanently locked in the contract. This not only risks permanent loss of user funds but also opens attack vectors, that malicious actors could exploit fake pools to intentionally trap tokens.

```
function executeMultiPath(address fromToken, uint256 fromTokenAmount, Utils.SinglePath[] calldata paths) internal {
    for (uint256 i = 0; i < paths.length; i++) {
        fromToken = i > 0 ? paths[i - 1].toToken : fromToken;

        // Record balance before swap (only if there are multiple paths)
        uint256 midTokenBalanceBefore;
        if (paths.length > 1 && i < paths.length - 1) {
            midTokenBalanceBefore = IERC20(paths[i].toToken).universalBalanceOf(address(this));
        }

        uint256 totalPercent;
        for (uint256 j = 0; j < paths[i].adapters.length; j++) {
            Utils.Adapter memory adapter = paths[i].adapters[j];
            require(whiteListAdapter[adapter.adapter], "Executor: adapter not whitelist");
            totalPercent += adapter.percent;
            uint256 fromAmountSlice = fromTokenAmount.decimalMul(adapter.percent);

            //DELEGATING CALL TO THE ADAPTER
            (bool success,) = adapter.adapter.delegatecall(
                abi.encodeWithSelector(
                    IAdapter.executeSimpleSwap.selector, fromToken, paths[i].toToken, fromAmountSlice, adapter.swaps
                )
            );
            if (success == false) {
                assembly {
                    let ptr := mload(0x40)
                    let size := returndatasize()
                    returndatacopy(ptr, 0, size)
                    revert(ptr, size)
                }
            }
        }
        require(totalPercent == SignedDecimalMath.ONE, "Executor: Invalid adapter total percent");

        // Calculate the actual amount received from this swap (only if there are multiple paths)
        if (paths.length > 1 && i < paths.length - 1) {
            uint256 midTokenBalanceAfter = IERC20(paths[i].toToken).universalBalanceOf(address(this));
            uint256 receivedAmount = midTokenBalanceAfter - midTokenBalanceBefore;
            fromTokenAmount = receivedAmount;
```

```
        }
    }
}
```

Listing 2.8: src/executor/Executor.sol

Impact This not only risks permanent loss of user funds but also opens attack vectors, that malicious actors could exploit fake pools to intentionally trap tokens.

Suggestion Refund the unspent `fromToken` to users. And add a function `sweep()` for users to sweep mid tokens.

Feedback from the project Currently, the param `minAmountOut` is used as protection, and this situation will not be considered.

2.2 Recommendation

2.2.1 Inconsistent check

Status Confirmed

Introduced by `Version 1`

Description The function `_validateSwapParams()` in the contract `DexSwap` requires that param `feeReceiver` cannot be the `DexSwap` contract's own address. However, the function `swap()` in the contract `Router` does not perform this check on the `feeReceiver` variable.

```
function swap(
    address fromToken,
    uint256 fromTokenAmount,
    address toToken,
    uint256 minAmountOut,
    bool feeOnFromToken,
    uint256 feeRate,
    address feeReceiver,
    Utils.MultiPath[] calldata paths
) external payable whenNotPaused nonReentrant {
    require(fromToken != toToken, "Router: fromToken and toToken cannot be the same");
    require(feeRate <= maxFeeRate, "Router: Fee Rate is too big");
    require(msg.value == (fromToken == UniversalERC20.ETH ? fromTokenAmount : 0), "Router:
        Incorrect msg.value");
    uint256 feeAmount;
```

Listing 2.9: src/router/Router.sol

```
function _validateSwapParams(SwapParams memory params, Adapter storage adapter) internal view
    {
    if (params.feeReceiver == address(this)) revert RouterError.IncorrectFeeReceiver();
    if (params.feeRate > maxFeeRate) revert RouterError.FeeRateTooBig();
    if (params.fromToken == params.toToken) revert RouterError.TokenPairInvalid();
    if (!adapter.isRegistered) revert RouterError.AdapterDoesNotExist();
    if (msg.value != (params.fromToken == UniversalERC20.ETH ? params.fromTokenAmount : 0)) {
        revert RouterError.IncorrectMsgValue();
    }
```

```
}
```

Listing 2.10: src/aggregatorRouter/DexSwap.sol

```
function swap(SwapParams memory params) external payable whenNotPaused nonReentrant {
    _swap(params);
}

function _swap(SwapParams memory params) internal {
    Adapter storage adapter = adapters[params.aggregatorId];

    // 1. check params
    _validateSwapParams(params, adapter);
```

Listing 2.11: src/aggregatorRouter/DexSwap.sol

Suggestion Add the check on the variable `feeReceiver` in the contract `Router`.

Feedback from the project This protocol's design pattern is not intended to grant admins permission to control the `feeReceiver` whitelist.

2.2.2 Non zero address checks

Status Confirmed

Introduced by `Version 1`

Description In the function `constructor()` of the contracts `Admin` and `WethAddress`, several address variables (e.g., `_admins`, `weth`) are not checked to ensure they are not zero addresses. It is recommended to add such checks to prevent potential mis-operations.

```
constructor(address[3] memory _admins) {
    admins = _admins;
}
```

Listing 2.12: src/library/Admin.sol

```
constructor(address weth) {
    WETH = weth;
}
```

Listing 2.13: src/executor/weth/WethAddress.sol

Suggestion Add non zero address checks accordingly.

2.2.3 Ensure proper checks on parameter `_admins`

Status Confirmed

Introduced by `Version 1`

Description The address elements in the array `_admins` should be unique.

```
constructor(address[3] memory _admins) {
    admins = _admins;
}
```

Listing 2.14: src/library/Admin.sol

Suggestion Add duplication checks accordingly.

2.2.4 Check the existence of the adapter before adding it

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `updateAdaptor()` in contract `Executor` allows the owner to add the address `_adapter` into the array `adapterList`. However, the function does not check the existence of the adapters, which allows the owner to add duplicate adapters.

```
function updateAdaptor(address _adapter, bool isAdd) external onlyOwner {
    whiteListAdapter[_adapter] = isAdd;
    if (isAdd) {
        adapterList.push(_adapter);
    } else {
        for (uint256 i = 0; i < adapterList.length; i++) {
            if (adapterList[i] == _adapter) {
                adapterList[i] = adapterList[adapterList.length - 1];
                adapterList.pop();
                break;
            }
        }
    }
}
```

Listing 2.15: src/executor/Executor.sol

Suggestion Check if the adapter exists before adding it.

2.2.5 Spelling and comment errors

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Misspelling of the recipient in contracts.

For example:

```
    address receipent,
```

Listing 2.16: src/aggregatorRouter/adapter/MagpieAdapter.sol

The comment "feeRate/10000" does not match the implementation (i.e., with 1e18 as denominator).

```
/// @param feeRate The fee rate to charge, feeRate/10000 will be charged to the feeReceiver
```

Listing 2.17: src/router/Router.sol

```
function chargeFee(address token, bool feeOnFromToken, uint256 amount, uint256 feeRate,
     address feeReceiver)
    internal
    returns (uint256, uint256)
{
    uint256 feeAmount = amount.decimalMul(feeRate);
    if (feeRate > 0) {
```

Listing 2.18: src/router/Router.sol

```
library SignedDecimalMath {
    int256 constant SignedONE = 10 ** 18;
    uint256 constant ONE = 1e18;

    function decimalMul(int256 a, int256 b) internal pure returns (int256) {
        return (a * b) / SignedONE;
    }

    function decimalDiv(int256 a, int256 b) internal pure returns (int256) {
        return (a * SignedONE) / b;
    }

    function abs(int256 a) internal pure returns (uint256) {
        return a < 0 ? uint256(a * -1) : uint256(a);
    }

    function decimalMul(uint256 a, uint256 b) internal pure returns (uint256) {
        return (a * b) / ONE;
    }

    function decimalDiv(uint256 a, uint256 b) internal pure returns (uint256) {
        return (a * ONE) / b;
    }

    function decimalRemainder(uint256 a, uint256 b) internal pure returns (bool) {
        return (a * ONE) % b == 0;
    }
}
```

Listing 2.19: src/library/SignedDecimalMath.sol

Suggestion Correct the spelling and ensure consistency between comments and implementation.

2.2.6 Redundant code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Both functions `receive()` in the contracts `Adapter1` are unnecessary. Since the execution logic involves the contract `Executor` making a delegatecall to `executeSimpleSwap()`

function of contract `Adapter1`, the `Executor` contract is the actual `recipient` that needs to receive `Ether`.

```
receive() external payable {}
```

Listing 2.20: src/adapter/mainnet/Adapter1.sol

```
receive() external payable {}
```

Listing 2.21: src/adapter/xDai/Adapter1.sol

Suggestion Remove the redundant code.

2.2.7 Add allowance revocation logic after swap operation

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `forceApprove()` is invoked to grant allowance to the adapters and executors. None of the implementations revokes the allowance after the swap operation completes. It is recommended to add allowance revocation logic after swap operation.

For example:

```
        // approve
        IERC20(fromToken).forceApprove(Kyber_Approver, fromTokenAmount);
```

Listing 2.22: src/aggregatorRouter/adapter/KyberAdapter.sol

Suggestion Add allowance revocation logic accordingly.

2.2.8 Replace the function `transfer()` with the function `call()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The functions `universalTransfer()` and `universalTransferFrom()` invoke the function `transfer()` to transfer Ether, which limits the gas cost to 2300 gas. If the receiver is a contract with complex logic in its `fallback` or `receive` functions, the transaction may fail due to the gas limitation.

```
function universalTransfer(IERC20 token, address payable to, uint256 amount) internal {
    if (amount > 0) {
        if (isETH(token)) {
            to.transfer(amount);
        } else {
            token.safeTransfer(to, amount);
        }
    }
}

function universalTransferFrom(IERC20 token, address from, address payable to, uint256 amount)
     internal {
    if (amount > 0) {
```

```
        if (isETH(token)) {
            to.transfer(amount);
        } else {
            token.safeTransferFrom(from, to, amount);
        }
    }
}
```

Listing 2.23: src/library/UniversalERC20.sol

Suggestion Replace the function `transfer()` with the function `call()`.

2.3 Note

2.3.1 Security consideration for delegatecall-based executor architecture

Introduced by `Version 1`

Description The contract `Executor` utilizes delegatecall to interact with adapters (e.g.,`Adapter1`), which inherits many other contracts (e.g., `MakerPsmExecutor`, `CurveV2Executor`). Currently, these delegate-called contracts only contain immutable variables, posing no security risk to the contract `Executor`'s state.

However, a critical risk emerges if future implementations:

1.Introduce new state variables to the implementation contracts (e.g., `MakerPsmExecutor`, `CurveV2Executor`).

2.These variables can be modified during `executeSimpleSwap` operations.

This could lead to malicious modification of the contract `Executor`'s state variables (e.g., `whiteListAdapter`, `adapterList`).

```
mapping(address => bool) public whiteListAdapter;
address[] public adapterList;
```

Listing 2.24: src/executor/Executor.sol

```
            (bool success,) = adapter.adapter.delegatecall(
                abi.encodeWithSelector(
                    IAdapter.executeSimpleSwap.selector, fromToken, paths[i].toToken,
                        fromAmountSlice, adapter.swaps
                )
            );
```

Listing 2.25: src/executor/Executor.sol

```
contract Adapter1 is
    IAdapter,
    AlgebraV3Executor,
    UniswapV2Executor,
    MakerPsmExecutor,
    CurveV1Executor,
    CurveV2Executor,
    UniswapV3Executor,
```

```
BalancerV1Executor,
BalancerV2Executor,
UniswapV3ForkExecutor,
VelodromeExecutor,
WethExecutor,
UniswapV4Executor
```

Listing 2.26: src/adapter/mainnet/Adapter1.sol

```
address public immutable dai;
```

Listing 2.27: src/executor/makerpsm/MakerPsmExecutor.sol

2.3.2 Security audit assumptions on trusted executors

Introduced by `Version 1`

Description The current security assessment of the protocol operates under the critical assumption that the external routers used in the adapter and executor contracts (i.e., `Adapter1` in the Ethereum mainnet, `Adapter1` in the Gnosis Chain, `UniAdapter`, `ParaswapAdapter`, `KyberAdapter`, `MagpieAdapter`, `MachaV2Adapter`, and `OneinchAdapter`) are fully trusted. This directly impacts all subsequent security evaluations and risk assessments conducted within the audit scope.

2.3.3 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `_admins`) can conduct sensitive operations, which introduces potential centralization risks. For example, the admin can pause the contract `Router`. 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 Weird ERC20 tokens

Introduced by `Version 1`

Description In this protocol, there are no whitelisting restrictions on ERC20 tokens. It is important to note that some weird ERC20 tokens (e.g., Fee-on-Transfer tokens) may result in unexpected behaviors. Specifically, if the `toToken` is a Fee-on-Transfer token, the actual amount received by the users may be inconsistent with the expected amount.

2.3.5 External pool/router without refund mechanism may cause unnecessary slippage

Introduced by `Version 1`

Description If the externally called router or pool lacks a refund mechanism after receiving the user's input tokens, any unused tokens (due to insufficient liquidity or partial execution) will become permanently locked, forcing users to bear unnecessary slippage on the swapped portion.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS