Security Audit Report for bgw-swap-aggregator-evm

Date: September 30, 2025 **Version:** 1.0
Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Bitget
Target	bgw-swap-aggregator-evm

Version History

Version	Date	Description
1.0	September 30, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-09-30 03:42:31+0000

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 bgw-swap-aggregator-evm of Bitget.

BWAggregator is a DEX aggregator that enables swaps across Uniswap and its fork protocols through specialized router and handler actions. The platform supports ERC20-to-ERC20, ETH-to-ERC20, and ERC20-to-ETH swaps through a meta-transaction architecture. Users can specify complex action sequences for the aggregator to execute atomically on their behalf, typically involving multiple swaps across different protocols. The system employs a three-phase execution method with fund tracking. Fee collection comprises two components: a fixed deduction amount and a proportional fee amount.

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

- contracts

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. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
bgw-swap-aggregator-evm	Version 1	0569e566d23e7d00ad17455de593fdeb279ccf75
	Version 2	3fce0d92ef279190906159d01b16b676ff6425cd

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.

[1] https://github.com/bitgetwallet/bgw-swap-aggregator-evm

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.

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 **six** potential security issues. Besides, we have **four** recommendations and **eight** notes.

- High Risk: 2
- Medium Risk: 1
- Low Risk: 3
- Recommendation: 4
- Note: 8

ID	Severity	Description	Category	Status
1	High	Incorrect pool verification in V3Handlers' swap callbacks	Security Issue	Fixed
2	High	Potential DoS risk in the function `managePoolManager()`	Security Issue	Fixed
3	Medium	Lack of access control in the function `handleFeeWithSign()`	Security Issue	Confirmed
4	Low	Potential DoS due to precision loss on `feeAmount`	Security Issue	Confirmed
5	Low	Ineffective gas optimization mechanism	Security Issue	Fixed
6	Low	Duplicate deduction logic in contract `BWAggPancakeV3Handler`	Security Issue	Fixed
7	-	Add address check during wrapping `Ether`	Recommendation	Fixed
8	-	Implement `_pool` address validation for Uniswap V2 pools	Recommendation	Confirmed
9	-	Avoid unused return variable	Recommendation	Fixed
10	-	Add explicit failure notification in the function `removeCallBack()`	Recommendation	Fixed
11	-	Ensure proper management of unspent ERC20 tokens	Note	-
12	-	Fee logic is controllable via parameters	Note	-
13	-	Potential risks for future integration	Note	-
14	-	Potential centralization risks	Note	-
15	-	Ensure the correctness of inputs in the backend	Note	-
16	-	Weird ERC20 Tokens	Note	-
17	-	Gas optimization retains 1 wei of tokens	Note	-
18	-	Router asset balance assumptions	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Incorrect pool verification in V3Handlers' swap callbacks

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contracts `BWAggPanacakeV3Handler` and `BWAggUniV3Handler`, the pool validation logic in functions `pancakeV3SwapCallback()` and `uniswapV3SwapCallback()` is incorrect. Specifically, the check passes when the initial condition fails by accepting any address that is not the legitimate factory-created pool. This allows any caller that is not a factory-created pool to circumvent the verification.

```
function uniswapV3SwapCallback(int256 amount0Delta, int256 amount1Delta, bytes calldata data)
     external override {
    require(amount0Delta > 0 || amount1Delta > 0, BWAggErrors.UniV3InvalidDelta());

    (address _pool, address actualActionTokenIn, address actualActionTokenOut, uint24 _feeRate)
        =
        abi.decode(data, (address, address, address, uint24));

    require(
        msg.sender == _pool || FACTORY.getPool(actualActionTokenIn, actualActionTokenOut,
            _feeRate) != msg.sender,
        BWAggErrors.UniV3CallbackNotPool()
    );

    bool zeroForOne = actualActionTokenIn < actualActionTokenOut;
    address settleToken;
    uint256 settleAmount;
    if (amount0Delta > 0) {
        // transfer token0 to pool, exactIn token0 is tokenIn
        settleToken = zeroForOne ? actualActionTokenIn : actualActionTokenOut;
        settleAmount = uint256(amount0Delta);
    } else {
        // transfer token1 to pool, exactIn token1 is tokenOut
        settleToken = zeroForOne ? actualActionTokenOut : actualActionTokenIn;
        settleAmount = uint256(amount1Delta);
    }

    BWAggTokenLib.transfer(settleToken, msg.sender, settleAmount);
}
```

Listing 2.1: contracts/bwAggregator/handler/v3Like/BWAggUniV3Handler.sol

```
function pancakeV3SwapCallback(int256 amount0Delta, int256 amount1Delta, bytes calldata data)
     external override {
    require(amount0Delta > 0 || amount1Delta > 0, BWAggErrors.UniV3InvalidDelta());

    (address _pool, address actualActionTokenIn, address actualActionTokenOut, uint24 _feeRate)
        =
        abi.decode(data, (address, address, address, uint24));

    require(
        msg.sender == _pool || FACTORY.getPool(actualActionTokenIn, actualActionTokenOut, _feeRate) != msg.sender,
        BWAggErrors.UniV3CallbackNotPool()
    );

    bool zeroForOne = actualActionTokenIn < actualActionTokenOut;
    address settleToken;
    uint256 settleAmount;
    if (amount0Delta > 0) {
        // transfer token0 to pool, exactIn token0 is tokenIn
        settleToken = zeroForOne ? actualActionTokenIn : actualActionTokenOut;
        settleAmount = uint256(amount0Delta);
    } else {
        // transfer token1 to pool, exactIn token1 is tokenOut
        settleToken = zeroForOne ? actualActionTokenOut : actualActionTokenIn;
        settleAmount = uint256(amount1Delta);
    }

    BWAggTokenLib.transfer(settleToken, msg.sender, settleAmount);
}
```

Listing 2.2: contracts/bwAggregator/handler/v3Like/BWAggPancakeV3Handler.sol

Impact This vulnerability could allow malicious actors to drain assets from the contracts by directly invoking the callback functions with crafted calldata.

Suggestion Revise the flawed logic to ensure only legitimate pools can call the callback.

2.1.2 Potential DoS risk in the function `managePoolManager()`

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `BWAggManager`, the function `managePoolManager()` is used to update the mapping `managerPoolManager`. However, the check on line 48 is incorrect. Specifically, the check allows state changes only for pool managers whose current status is `false`, resulting in a DoS issue during deactivation for an active manager.

```
42  function managePoolManager(bytes4 selector, address poolManager, bool isActive) external
        onlyOwner {
43      _checkAddress(poolManager);
44      _managePoolManager(selector, poolManager, isActive);
45  }
46
47  function _managePoolManager(bytes4 selector, address poolManager, bool isActive) internal
        onlyOwner {
48      require(!managerPoolManager[selector][poolManager], IBWSwapBase.ValueCanNotBeEqual());
49      managerPoolManager[selector][poolManager] = isActive;
```

```
        emit ManagePoolManager(selector, poolManager, isActive);
    }
```

Listing 2.3: contracts/bwAggregator/BWAggManager.sol

Impact This flaw may prevent the deactivation of active pool managers.

Suggestion Revise the code logic accordingly.

2.1.3 Lack of access control in the function `handleFeeWithSign()`

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description In the contract `BWSwapFee`, the function `handleFeeWithSign()` validates the signature and consumes the nonce. However, the function does not apply any access control. This allows arbitrary callers to front-run a legitimate swap attempt via the function. Once the nonce is consumed, the legitimate swap transaction reverts.

```
    function handleFeeWithSign(
        address _msgSender,
        uint256 _deductAmount,
        uint256 _deductToIndex,
        uint256 _nonce,
        uint256 _deadline,
        bytes calldata _signature
    ) external returns (HandleFeeCallback memory callback) {
        FeeConfig memory tempConfig = config;

        (callback.isNeedDeduct, callback.deductTo) = _checkDeduct(tempConfig, _deductAmount,
            _deductToIndex);
        if (tempConfig.isNeedCheckSigner) {
            _checkSigner(_msgSender, _nonce, _deadline, _signature, tempConfig.signerArray);
        }
    }
```

Listing 2.4: contracts/bwCommon/bwSwapFee/BWSwapFee.sol

Impact Malicious actors can deliberately cause swaps to fail, thereby disrupting the project's swap functionality.

Suggestion Add a check to ensure the `msg.sender` is a trusted caller.

Feedback from the project There are no financial incentives for attackers. If it occurs, it would only impact free-type transactions with signature verification enabled. This can be mitigated by disabling signature checks (set `isNeedCheckSigner` to `false`). For large-scale attacks, comprehensive defenses are available, including private transaction pools and re-deploying fee contracts with enhanced caller verification.

2.1.4 Potential DoS due to precision loss on `feeAmount`

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the contract `BWAggCollectLib`, the function `collectFee()` calculates fees using integer division that rounds down. When `amountForFee * feeRate < feeRateBase`, the division result will be 0, causing the `require(feeAmount>0)` check to fail and revert the entire transaction. This is problematic for tokens with low decimals but high values, where reasonable swap amounts may not generate sufficient fees to pass the check.

```
function collectFee(
    bool feeTokenIsETH,
    bool isFirstCollectFee,
    uint256 amountForFee,
    uint256 feeRate,
    uint256 feeRateBase,
    address feeToken,
    address feeTo
) internal returns (uint256 feeAmount) {
    feeAmount = (amountForFee * feeRate) / feeRateBase;
    require(feeAmount > 0, BWAggErrors.FeeAmountCanNotBeZero());

    if (feeTokenIsETH) {
        BWAggTokenLib.transfer(address(0), feeTo, feeAmount);
    } else {
        if (isFirstCollectFee) {
            BWAggTokenLib.transferFrom(feeToken, msg.sender, feeTo, feeAmount);
        } else {
            // TODO: Handle high-value tokens, refer to audit & server recommendations
            BWAggTokenLib.transferSave1Wei(feeToken, feeTo, feeAmount);
        }
    }
}
```

Listing 2.5: contracts/bwAggregator/libs/BWAggCollectLib.sol

Impact This rounding mechanism leads to potential DoS for small-value transactions.

Suggestion Implement proper fee calculation that rounds up or adjusts the zero fee handling logic.

Feedback from the project The project stated that a minimum swap amount check will be implemented on the frontend for fee-related transactions. For amounts below the minimum threshold, the project will either block the transaction or adjust the fee-charging direction.

2.1.5 Ineffective gas optimization mechanism

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contracts `BWAggUniV3Handler`, `BWAggPancakeV3Handler`, and `BWAggUniV4Handler`, the gas-saving logic contains a flawed check that prevents the 1-wei saving mechanism from working as intended. Specifically, the check compares `unspentAmount` (the balance of `_tokenIn` at the `_swapTo` address) against `amountIn` (the balance of `_tokenIn` at the current contract). Since these values generally represent balances at different addresses, the likelihood of this condition being met is very low, making the gas optimization ineffective.

```
    // 2.pay tokenIn
    uint256 amountIn = BWAggTokenLib.getBalance(tokenIn, address(this));
    uint256 unspentAmount = BWAggTokenLib.getBalance(tokenIn, _swapTo);
    if (unspentAmount == amountIn && amountIn > 1) {
        amountIn = amountIn - BWAggConstants.SAVE_1_WEI; // 1 wei gas saving
    }
```

Listing 2.6: contracts/bwAggregator/handler/v4Like/BWAggUniV4Handler.sol

```
    uint256 balanceBefore = BWAggTokenLib.getBalance(_tokenOut, _swapTo);
    uint256 unspentAmount = BWAggTokenLib.getBalance(_tokenIn, _swapTo);
    if (unspentAmount == amountIn && amountIn > 1) {
        amountIn = amountIn - BWAggConstants.SAVE_1_WEI; // 1 wei gas saving
    }
```

Listing 2.7: contracts/bwAggregator/handler/v3Like/BWAggUniV3Handler.sol

```
    uint256 balanceBefore = BWAggTokenLib.getBalance(_tokenOut, _swapTo);
    uint256 unspentAmount = BWAggTokenLib.getBalance(_tokenIn, _swapTo);
    if (unspentAmount == amountIn && amountIn > 1) {
        amountIn = amountIn - BWAggConstants.SAVE_1_WEI; // 1 wei gas saving
    }
```

Listing 2.8: contracts/bwAggregator/handler/v3Like/BWAggPancakeV3Handler.sol

Impact The gas optimization mechanism is rarely triggered due to the flawed check.

Suggestion Revise the logic accordingly.

2.1.6 Duplicate deduction logic in contract `BWAggPancakeV3Handler`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `BWAggPancakeV3Handler`, the swap amount is designed to decrease by 1 wei to save gas when calling `swap()` on the Pancake pool. However, it implements a duplicate deduction logic in lines 49 and 55. This could lead to fund loss (1 wei) or transaction revert in edge cases where `amountIn` is reduced to zero.

```
48      if (unspentAmount == amountIn && amountIn > 1) {
49          amountIn = amountIn - BWAggConstants.SAVE_1_WEI; // 1 wei gas saving
50      }
51
52      IUniswapV3Pool(_pool).swap(
53          _swapTo,
54          zeroForOne,
55          int256(amountIn - 1), // save 1 wei gas
```

```
            sqrtPriceLimitX96,
            abi.encode(_pool, _tokenIn, _tokenOut, feeRate)
        );
```

Listing 2.9: contracts/bwAggregator/handler/v3Like/BWAggPancakeV3Handler.sol

Meanwhile, the implementation is inconsistent with the implementation in `BWAggUniV3Handler`.

```
        IUniswapV3Pool(_pool).swap(
            _swapTo, zeroForOne, int256(amountIn), sqrtPriceLimitX96, abi.encode(_pool, _tokenIn,
                _tokenOut, feeRate)
        );
```

Listing 2.10: contracts/bwAggregator/handler/v3Like/BWAggUniV3Handler.sol

Impact This implementation leads to consistent 1 - wei fund loss per transaction and potential DoS in edge cases.

Suggestion Remove redundant 1 - wei deduction to align with the contract `BWAggUniV3Handler` implementation.

2.2 Recommendation

2.2.1 Add address check during wrapping `Ether`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `BWAggregatorRouter`, the function `_executeAction()` invokes the function `wrapETH()` when the action type equals`ROUTER_WRAP`. The function `wrapETH()` transfers the input `actionTokenInAmount` of `Ether` to the custom target address. However, in the execution flow, these functions do not validate whether the `action.tokenIn` equals `address(0)`. As a result, a user could specify the action type as `ROUTER_WRAP` to use the contract's `Ether`. It is recommended to add a validation for the input token.

```
    function _executeAction(SwapAction calldata action, uint256 actionTokenInAmount, address
         originSwapTokenIn)
        internal
        returns (uint256 actionSwappedAmount)
    {
        actionTokenInAmount =
            BWAggTokenLib.calculateSave1WeiAmount(action.tokenInSource, action.tokenIn,
                actionTokenInAmount);
        if (action.actionType == ActionType.ROUTER_WRAP) {
            actionSwappedAmount = BWAggWrapLib.wrapETH(action.tokenOut, actionTokenInAmount, action
                .tokenOutTarget);
        } else if (action.actionType == ActionType.ROUTER_UN_WRAP) {
            BWAggTokenLib.transferActionTokenIn(
                msg.sender, action.tokenInSource, action.tokenIn, address(this),
                    actionTokenInAmount, originSwapTokenIn
            );
            actionSwappedAmount = BWAggWrapLib.unwrapWETH(action.tokenIn, actionTokenInAmount,
                action.tokenOutTarget);
        } else if (action.actionType == ActionType.ROUTER_UNI_V2) {
```

Listing 2.11: contracts/bwAggregator/BWAggregatorRouter.sol

```
function wrapETH(address weth, uint256 amount, address to) internal returns (uint256) {
    require(amount > 0, BWAggErrors.WrapTokenAmountCanNotZero());
    IWETH9(weth).deposit{value: amount}();
    if (to != address(this)) {
        BWAggTokenLib.transfer(weth, to, amount);
    }
    return amount;
}
```

Listing 2.12: contracts/bwAggregator/libs/BWAggWrapLib.sol

Suggestion Add a check to validate the input token.

2.2.2 Implement `_pool` address validation for Uniswap V2 pools

Status Confirmed

Introduced by `Version 1`

Description Since the `_pool` address is derived from user-provided `SwapAction` parameters in the main router contract, an attacker can supply any arbitrary contract address that implements the `IUniswapV2Pair` interface. This could potentially lead to unauthorized external contract calls. It is recommended to implement pool address validation using the Uniswap V2 factory pattern to mitigate this risk.

```
            IUniswapV2Pair(_pool).swap(0, amountOut, _swapTo, new bytes(0));
```

Listing 2.13: contracts/bwAggregator/libs/BWAggUniswapLib.sol

```
            IUniswapV2Pair(_pool).swap(0, amountOut, _swapTo, new bytes(0));
```

Listing 2.14: contracts/bwAggregator/handler/v2Like/BWAggUniV2Handler.sol

```
            IUniswapV2Pair(_pool).swap(amountOut, 0, _swapTo, new bytes(0));
```

Listing 2.15: contracts/bwAggregator/handler/v2Like/BWAggUniV2Handler.sol

Suggestion Implement pool address validation using the Uniswap V2 factory pattern.

Feedback from the project The project does not plan to implement this as it can incur additional gas costs.

2.2.3 Avoid unused return variable

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `BWAggCollectLib`, the function `collectDeduct()` declares `deductAmount_` as a named return variable. However, the function uses an explicit return statement without assignment to the variable `deductAmount_`.

```
function collectDeduct(
    bool isNeedDeduct,
    bool feeTokenIsETH,
    bool isFirstCollectDeduct,
    address feeToken,
    address deductTo,
    uint256 deductAmount
) internal returns (uint256 deductAmount_) {
    if (isNeedDeduct) {
        if (feeTokenIsETH) {
            BWAggTokenLib.transfer(address(0), deductTo, deductAmount);
        } else {
            if (isFirstCollectDeduct) {
                BWAggTokenLib.transferFrom(feeToken, msg.sender, deductTo, deductAmount);
            } else {
                BWAggTokenLib.transfer(feeToken, deductTo, deductAmount);
            }
        }
        return deductAmount;
    }
}
```

Listing 2.16: contracts/bwAggregator/libs/BWAggCollectLib.sol

Suggestion Use or remove any unused return variables.

2.2.4 Add explicit failure notification in the function `removeCallBack()`

Status Fixed in Version 2

Introduced by Version 1

Description In the contract `BWAggManager`, the function `removeCallBack()` is designed to remove a specified factory address from the callback list associated with a given function selector. However, the current implementation fails to provide explicit notification when the specified factory address is not found in the list, leaving callers unable to determine operation success.

```
function removeCallBack(bytes4 selector, address factory) external onlyOwner {
    address[] storage factories = managerCallback[selector];
    for (uint256 i = 0; i < factories.length; i++) {
        if (factories[i] == factory) {
            factories[i] = factories[factories.length - 1];
            factories.pop();
            emit ManageCallBack(selector, factory, false);
            break;
        }
    }
}
```

Listing 2.17: contracts/bwAggregator/BWAggManager.sol

Suggestion Implement revert with a descriptive error message when the factory address is not found in the callback list.

2.3 Note

2.3.1 Ensure proper management of unspent ERC20 tokens

Introduced by `Version 1`

Description Uniswap V3 and V4 compatible swaps may not fully consume the specified input amounts due to factors such as insufficient liquidity, slippage constraints, or market conditions. When this occurs, unspent tokens are retained in the contract `BWAggregatorRouter` instead of being refunded to users.

This creates a potential attack vector where malicious actors could extract these residual funds via swaps. To mitigate this risk, the project should implement proper swap parameter optimization and continuous balance monitoring to prevent significant token accumulation. This approach makes any potential value extraction economically unprofitable.

Feedback from the project By design, any unspent intermediate token balances during swaps remain in the contract `BWAggregatorRouter`, which will be monitored post-launch. Users experience no actual loss as they only require receiving amounts that meet their specified `minAmountOut` with slippage tolerance. Unspent intermediate tokens are typically dust amounts where transfer costs exceed token value. For the larger amounts, it can be recovered via functions `rescueETH()` and `rescueERC20()`.

2.3.2 Fee logic is controllable via parameters

Introduced by `Version 1`

Description The fee logic within the contract `BWAggregatorRouter` is directly influenced by the unverified `swapType` parameter supplied in the parameter. Users can specify the `swapType` as `SwapType.FREE` to entirely circumvent the standard fee charging mechanism. This circumvention is effective when the global configuration parameter `config.isNeedCheckSigner` is set to `false`.

```
        if (_params.swapType == SwapType.FREE) {
            receivedAmount = _swapForFree(_feeCallback, _params, _actions);
        } else if (_params.swapType == SwapType.ETH_TOKEN) {
            (feeAmount, receivedAmount) = _swapETH2Token(false, _feeCallback, _params, _actions);
        } else if (_params.swapType == SwapType.TOKEN_ETH) {
            (feeAmount, receivedAmount) = _swapToken2ETH(false, _feeCallback, _params, _actions);
        } else if (_params.swapType == SwapType.TOKEN_WHITE) {
            (feeAmount, receivedAmount) = _swapToken2White(false, _feeCallback, _params, _actions);
        } else if (_params.swapType == SwapType.TOKEN_TOKEN) {
            (feeAmount, receivedAmount) = _swapToken2Token(false, _feeCallback, _params, _actions);
        }
```

Listing 2.18: contracts/bwAggregator/BWAggregatorRouter.sol

```
    function _swapForFree(
        IBWSwapFee.HandleFeeCallback memory _feeCallback,
        SwapParams calldata _params,
        SwapAction[] calldata _actions
    ) internal returns (uint256 receivedAmount) {
        bool tokenInIsETH = _params.tokenIn == address(0);
        bool tokenOutIsETH = _params.tokenOut == address(0);
        if (tokenInIsETH && !tokenOutIsETH) {
            (, receivedAmount) = _swapETH2Token(true, _feeCallback, _params, _actions);
        } else if (!tokenInIsETH && tokenOutIsETH) {
            (, receivedAmount) = _swapToken2ETH(true, _feeCallback, _params, _actions);
        } else {
            (, receivedAmount) = _swapToken2Token(true, _feeCallback, _params, _actions);
        }
        return receivedAmount;
    }
```

Listing 2.19: contracts/bwAggregator/BWAggregatorRouter.sol

Feedback from the project This is by design.

2.3.3 Potential risks for future integration

Introduced by `Version 1`

Description Handler contracts in the v2Like, v3Like, and v4Like folders are audited and serve as templates for future integrations. To integrate with protocols that differ from Uniswap, new handlers may be developed based on these templates. However, these newly developed handlers may introduce additional risks and should be approached with caution before deployment.

2.3.4 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., the owner role) can conduct sensitive operations, which introduces potential centralization risks. For example, the contract `BWAggregatorRouter`'s owner can designate an operator who can pause the swap functionality or rescue tokens from the contract. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

Feedback from the project The project uses multi-signature to control privileged roles instead of EOAs.

2.3.5 Ensure the correctness of inputs in the backend

Introduced by `Version 1`

Description The swapping results of the contract `BWAggregatorRouter` are highly dependent on the provided inputs. Critical inputs that require careful construction include, but are not limited to, slippage tolerances, swap input amounts, and swap paths. The project team should ensure that the correct inputs are constructed in the backend. Improper construction may result in transaction reverts or financial losses.

Feedback from the project The backend ensures parameter correctness, while the contract only validates the `minAmountOut` parameter.

2.3.6 Weird ERC20 Tokens

Introduced by `Version 1`

Description In this project, there are no whitelisting restrictions on swapped ERC20 tokens. It is important to note that some weird ERC20 tokens (e.g., Fee-on-Transfer tokens) are not supported and may result in unexpected behaviors.

Feedback from the project This is by design. The project allows arbitrary token swap. Some special token swaps may fail, which is expected behavior given the partial support for specialized token types.

2.3.7 Gas optimization retains 1 wei of tokens

Introduced by `Version 1`

Description The project implements a gas optimization mechanism. Specifically, the router and handler contracts retain 1 wei of each ERC20 token after its first swap. Once the 1 wei balance is established, subsequent swaps for the same token operate with full precision without additional reductions.

This avoids expensive zero-to-non-zero storage transitions in subsequent operations involving the same token. The mechanism reduces transaction costs as modifying existing nonzero storage values costs significantly less than initializing zero values.

This optimization creates a minor discrepancy where users or the fee collector receive slightly fewer tokens than expected. For most ERC20 tokens, 1 wei represents a negligible value. However, the impact may be more noticeable with low-decimal or high-value tokens.

Feedback from the project This is by design. The project states that the slippage check (i.e., `receivedAmount >= _params.minAmountOut`) addresses this minor discrepancy.

2.3.8 Router asset balance assumptions

Introduced by `Version 1`

Description The design of the contract `BWAggregatorRouter` assumes it should not hold any token balances. Furthermore, the balance verification is only performed when the `originTokenIn` is `Ether`. Therefore, the change in the `BWAggregatorRouter`'s token balances before and after swap execution is not considered during the audit.

Feedback from the project The project stated that the contract `BWAggregatorRouter` does not hold currency balances. The balance verification is specifically implemented when `originTokenIn` is `Ether`.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS