BLOCKSEC

Security Audit

Report for bgw-swap-aggregator-evm

Date: May 18, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

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

Version History

Version	Date	Description
1.0	May 18, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-05-19 07:01:44+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 the Audit Target

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

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of bgw-swap-aggregator-evm of bitgetwallet.

The Bitget Aggregator EVM project introduces `BWAggFeeHandler`, a new contract enabling a fee distribution mechanism via the standard handler interface (i.e., `IBWAggBaseHandler.swap()`). On each invocation, the function decodes the fee parameters, optionally charges a flat fee, and splits a percentage fee between a share recipient and the platform treasury. Before any distribution takes place, an ECDSA signature from an authorized signer is verified, ensuring that only approved fee-split parameters can be settled on-chain.

Note this audit only focuses on the smart contracts in the following directories/files. Code prior to and including the baseline version 0, where applicable, is outside the scope of this audit and assumes to be reliable and secure.

- contracts/bwAggregator/shareFee/BWAggFeeHandler.sol
- contracts/bwAggregator/shareFee/IBWAggFeeHandler.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the Target 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 0`	`6b3c00c8558382c2f4c5d3a1c0b3688b67a40895`
	`Version 1`	`333f07b0371178db8aea0d1c6088149a1271ad42`
	`Version 2`	`592c799d17732d13f2ed788464eef3a3781ad16c`

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,

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

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 Target, 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 the Target.

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 the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target 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 **one** potential security issue. Besides, we have **three** recommendations and **three** notes.

- Low Risk: 1
- Recommendation: 3
- Note: 3

ID	Severity	Description	Category	Status
1	Low	Lack of check on token consistency	Security Issue	Fixed
2	-	Use the received amount as input	Recommendation	Confirmed
3	-	Enforce `feeRate` bounds in function `swap()`	Recommendation	Confirmed
4	-	Bind `tokenIn` in the signed message	Recommendation	Fixed
5	-	Potential centralization risks	Note	-
6	-	Trusted off-chain logic	Note	-
7	-	Weird ERC20 Tokens	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Lack of check on token consistency

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `BWAggFeeHandler`, function `swap()` deducts fees from `tokenIn` and returns the remaining amount. Contract `BWAggregatorRouter` records this returned value for `tokenOut`. Thus, correct accounting requires `tokenIn` to be the same as `tokenOut`. However, it does not enforce the requirement. When a user submits an action with `tokenIn != tokenOut`, the router's accounting becomes inconsistent with the actual execution.

```
26  function swap(
27      address _swapTo,
28      address _tokenIn,
29      address,
30      /*_tokenOut*/
31      address,
32      /*_pool*/
33      bytes memory _extraData
34  )
35      external
36      payable
37      override
38      onlyCaller
39      returns (uint256 swappedAmount)
```

```
40  {
41      (
42          IBWAggregatorRouter.FeeParams memory feeParams,
43          ShareRecipient memory share,
44          ShareFeeSignParams memory signParams
45      ) = abi.decode(_extraData, (IBWAggregatorRouter.FeeParams, ShareRecipient,
            ShareFeeSignParams));
```

Listing 2.1: contracts/bwAggregator/shareFee/BWAggFeeHandler.sol

```
288 function _executeSwap(SwapParams calldata _params, SwapAction[] calldata _actions, uint256
        ctxAmountIn) internal {
289     uint256 actionsLength = _actions.length;
290     uint256 maxActionTokenCount = _params.totalTokenCount;
291
292     ActionTokenState[] memory actionTokenStates = new ActionTokenState[](maxActionTokenCount);
293     uint256 actionTokenCount = 1;
294
295     // initialize the first state
296     actionTokenStates[0] =
297         ActionTokenState({tokenAddress: _params.tokenIn, totalAmount: ctxAmountIn, usedAmount:
                0, usedPercent: 0});
298
299     for (uint256 i = 0; i < actionsLength; i++) {
300         SwapAction calldata action = _actions[i];
301
302         require(
303             action.percent != 0 && action.percent <= BWAggConstants.PERCENT_BASE, BWAggErrors.
                    ActionInvalidPercent()
304         );
305
306         uint256 actionTokenInAmount = _handleActionTokenIn(actionTokenStates, action,
                actionTokenCount);
307         uint256 actionSwappedAmount = _executeAction(action, actionTokenInAmount, _params.
                tokenIn);
308         actionTokenCount = _handleActionTokenOut(
309             actionTokenStates, action, actionSwappedAmount, actionTokenCount,
                    maxActionTokenCount
310         );
311     }
312     // check all token states
313     _checkActionTokenState(actionTokenStates, actionTokenCount, _params.tokenOut);
314 }
```

Listing 2.2: contracts/bwAggregator/BWAggregatorRouter.sol

```
358 function _handleActionTokenOut(
359     ActionTokenState[] memory actionTokenStates,
360     SwapAction calldata action,
361     uint256 swappedAmount,
362     uint256 actionTokenCount,
363     uint256 maxActionTokenCount
364 ) internal view returns (uint256 newTokenCount) {
```

```
365       bool isInRouter = action.tokenOutTarget == address(this);
366       if (isInRouter) {
367           for (uint256 i = 0; i < actionTokenCount; i++) {
368               if (actionTokenStates[i].tokenAddress == action.tokenOut) {
369                   actionTokenStates[i].totalAmount += swappedAmount;
370                   return actionTokenCount;
371               }
372           }
```

Listing 2.3: contracts/bwAggregator/BWAggregatorRouter.sol

Impact This vulnerability may cause incorrect accounting in the contract `BWAggregatorRouter`.

Suggestion Add a check to ensure `tokenIn == tokenOut`.

2.2 Recommendation

2.2.1 Use the received amount as input

Status Confirmed

Introduced by `Version 1`

Description In contract `BWAggFeeHandler`, function `swap()` uses the contract's total `_tokenIn` balance as the input amount. If the contract already holds the same token, these tokens are incorrectly treated as part of the current swap's input. It is recommended to use only the received amount from the caller.

```
52        uint256 balance = BWAggTokenLib.getBalance(_tokenIn, address(this));
53        require(balance > 0, InvalidAmount());
```

Listing 2.4: contracts/bwAggregator/shareFee/BWAggFeeHandler.sol

Suggestion Revise the logic accordingly.

Feedback from the project This behavior is by design, and contract `BWAggFeeHandler` does not hold currency balances.

2.2.2 Enforce `feeRate` bounds in function `swap()`

Status Confirmed

Introduced by `Version 1`

Description In contract `BWAggFeeHandler`, function `swap()` uses `feeParams.feeRate` to compute the fee split without validating it against the configured bounds (i.e., `feeRateMin` and `feeRateMax`). As a result, an out-of-range rate can be accepted on-chain.

```
52        uint256 balance = BWAggTokenLib.getBalance(_tokenIn, address(this));
53        require(balance > 0, InvalidAmount());
```

Listing 2.5: contracts/bwAggregator/shareFee/BWAggFeeHandler.sol

Suggestion Add a fee-rate bounds check using the configured `feeRateMin` and `feeRateMax` before computing `feeAmount`.

Feedback from the project This behavior is by design.

2.2.3 Bind `tokenIn` in the signed message

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `BWAggFeeHandler`, function `_verifyShareFeeSign()` verifies the signature over the fee-splitting parameters applied to `tokenIn`. However, the signed `msgHash` does not bind the `tokenIn`. As a result, the fee signature could be applied to unexpected tokens.

```
115       bytes32 msgHash = keccak256(
116           abi.encode(
117               feeRate,
118               share.recipient,
119               share.shareRate,
120               signParams.nonce,
121               signParams.deadline,
122               block.chainid,
123               address(this)
124           )
125       );
```

Listing 2.6: contracts/bwAggregator/shareFee/BWAggFeeHandler.sol

Suggestion Include `tokenIn` in the signed message.

2.3 Note

2.3.1 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., the `owner`, `operator`, and the array `signerArray`) can perform sensitive operations, which introduces potential centralization risks. For example, the owner can whitelist arbitrary handlers, replace the fee contract, and rewrite the fee configuration. The operator can pause core functionality or rescue tokens. The trusted `signerArray` and the handler `caller` allowlist are centrally managed by the `owner`, meaning fee splitting and distribution rely on privileged key security. If the private keys of these privileged accounts are lost or maliciously exploited, or if these trusted roles behave improperly, it could pose significant risks to the protocol.

2.3.2 Trusted off-chain logic

Introduced by `Version 1`

Description The project depends on the correctness and consistency of the inputs constructed off-chain. Critical inputs, such as slippage tolerances, swap input amounts, swap paths, action sequences, fee parameters, and the corresponding signature payloads, should be carefully constructed. Improper construction may result in transaction reverts or financial losses.

2.3.3 Weird ERC20 Tokens

Introduced by `Version 1`

Description In this project, there are no whitelisting restrictions on ERC20 tokens, and the return value of the function `swap()` only represents the amount transferred out, not the amount actually received. 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.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS