BLOCKSEC

Security Audit
Report for Dispatcher Contracts

Date: July 17, 2026 **Version:**

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Bitget
Target	Dispatcher Contracts

Version History

Version	Date	Description
	July 17, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-07-20 07:54:54+0000

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 Dispatcher Contracts of Bitget.

The Dispatcher Contracts of Bitget is a payment infrastructure, which serves for two business flows: Checkout payments and Business call dispatch. Specifically, checkout payments pull a user's tokens, and distribute them to payout recipients within optional swaps through an allowlisted aggregator. As for business call dispatch, it pulls the user's funds and executes users' intended calls. It is worth noting that the whitelist for calls targets is per business type and can be circumvented by the owner via the unrestricted flag.

Note this audit focuses on the smart contracts located in the src/ directory, excluding the following directories/files:

- src/interface

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
bgwdispatcher	`Version 1`	`6478c8f1167b0d3fe9d0f7aaa27ced195721e205`
	`Version 2`	`ab20b4a035e335b572035f6661afee69e1142c76`

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/bgwdispatcher

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 **two** recommendations and **two** notes.

- Low Risk: 1
- Recommendation: 2
- Note: 2

ID	Severity	Description	Category	Status
1	Low	Lack of allowance revocation after swaps	Security Issue	Confirmed
2	-	Remove redundant code	Recommendation	Fixed
3	-	Validate the total native value before executing `calls[]`	Recommendation	Fixed
4	-	Potential centralization risks	Note	-
5	-	Proper configuration for unrestricted business calls	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Lack of allowance revocation after swaps

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the contract `BGWDispatcher`, the function `_executeSwap()` delegates the token approval to the swap target but never resets it to zero after the swap. If the swap target does not fully consume the allowance, the residual persists. As a result, this may allow others to use the residual allowance from the contract (e.g., via the function `dispatch()`), leading to potential loss of funds.

```
353   function _executeSwap(address tokenIn, uint256 amountIn, SwapCallData calldata swapCallData)
          internal {
354       address target = swapCallData.target;
355       require(isSwapTarget(target), SwapTargetNotAllowed());
356       TokenHelper.safeApproveWithRetry(tokenIn, target, amountIn);
357       // Reentrancy guard: entrypoints are nonReentrant and nonce is consumed first.
358       target.functionCall(swapCallData.data);
359   }
```

Listing 2.1: src/BGWDispatcher.sol

```
38   function safeApproveWithRetry(address token, address to, uint256 value) internal {
39       (bool success, bytes memory data) = token.call(abi.encodeWithSelector(APPROVE_SELECTOR, to,
             value));
40       if (!(success && (data.length == 0 || abi.decode(data, (bool))))) {
```

```
41          safeApprove(token, to, 0);
42          safeApprove(token, to, value);
43      }
44  }
```

Listing 2.2: src/libs/TokenHelper.sol

```
70  function setSwapTarget(address _target, bool _allowed) external onlyOwner {
71      _checkZeroAddress(_target);
72      require(_allowed ? _swapTargetSet.add(_target) : _swapTargetSet.remove(_target),
            ValueCanNotBeEqual());
73      emit SwapTargetUpdated(_target, _allowed);
74  }
```

Listing 2.3: src/base/BGWDispatcherStorage.sol

Impact Potential loss of funds due to the lack of approval revocation after swaps.

Suggestion Revise the code accordingly.

Feedback from the project The project stated that all swap targets are trusted third-party contracts.

2.2 Recommendation

2.2.1 Remove redundant code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the interface `IErrors`, the error `NativeTokenMismatch()` is defined but never used. It is recommended to remove this unused error.

```
45  /// @notice The token does not match the NATIVE sentinel.
46  error NativeTokenMismatch();
```

Listing 2.4: src/interface/IErrors.sol

Suggestion Remove the unused error.

2.2.2 Validate the total native value before executing `calls[]`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `for` loop of the function `_execute()`, the total used native value (i.e., `totalValue`) is checked (i.e., `<= allowedValue`) before each call. It is recommended to validate the total used value in a separate loop for gas optimization.

```
234  function _execute(Call[] calldata calls, uint16 businessType, uint256 allowedValue) internal {
235      uint256 len = calls.length;
236      require(len != 0, BadCalls());
237      uint256 totalValue;
```

```
238         for (uint256 i; i < len; ++i) {
239             _checkZeroAddress(calls[i].target);
240             require(_isBusinessCallAllowed(businessType, calls[i].target), TargetNotAllowed());
241             totalValue += calls[i].value;
242             require(totalValue <= allowedValue, NativeValueMismatch());
243             calls[i].target.functionCallWithValue(calls[i].data, calls[i].value);
244         }
245     }
```

Listing 2.5: src/BGWDispatcher.sol

Suggestion It is recommended to validate the total native value of the entire batch before any call execution.

2.3 Note

2.3.1 Potential centralization risks

Introduced by Version 1

Description In this project, several privileged roles (e.g., `operator`) can conduct sensitive operations, which introduce potential centralization risks. For example, the role `operator` can arbitrarily pause or unpause the protocol. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.2 Proper configuration for unrestricted business calls

Introduced by Version 1

Description In the function `_execute()`, the business calls marked as unrestricted (i.e., the flag `unrestricted` is set to `true`) allow users to perform arbitrary calls such as `token.transferFrom()`. Once the business signer (i.e., `_businessConfig[businessType].signers`) approves such calls, users are able to drain approved assets via the function `dispatch()`. The project must ensure proper configuration for unrestricted business calls to prevent potential losses.

```
234     function _execute(Call[] calldata calls, uint16 businessType, uint256 allowedValue) internal {
235         uint256 len = calls.length;
236         require(len != 0, BadCalls());
237         uint256 totalValue;
238         for (uint256 i; i < len; ++i) {
239             _checkZeroAddress(calls[i].target);
240             require(_isBusinessCallAllowed(businessType, calls[i].target), TargetNotAllowed());
241             totalValue += calls[i].value;
242             require(totalValue <= allowedValue, NativeValueMismatch());
243             calls[i].target.functionCallWithValue(calls[i].data, calls[i].value);
244         }
245     }
```

Listing 2.6: src/BGWDispatcher.sol

```
147   function _isBusinessCallAllowed(uint16 businessType, address target) internal view returns (
          bool) {
148       BusinessConfig storage config = _businessConfig[businessType];
149       return config.unrestricted || config.targets.contains(target);
150   }
```

Listing 2.7: src/base/BGWDispatcherStorage.sol


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS