BLOCKSEC

Security Audit
Report for BGW7702

Date: April 3, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Bitget
Target	BGW7702

Version History

Version	Date	Description
1.0	April 3, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-04-03 03:56:34+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 BGW7702 of Bitget.

BGW7702 is a smart-account framework built for EIP-7702 delegation with optional ERC-4337 compatibility. It separates execution and control planes: BGW7702Logic handles batched call execution (aggregate/aggregateV2), while BGW7702Admin manages EIP-712 dual-signature validation (platform signer + account owner) and unordered nonce tracking. BGW7702Base provides ownership, operator-based pause/rescue controls, and shared admin utilities, and BGW7702Helper maintains business-type metadata mappings. The design aims to keep execution logic lightweight and centralize sensitive state management in a dedicated admin contract, enabling operational control, emergency pause, and consistent signature policy enforcement across delegated accounts.

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 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
BGW7702	`Version 1`	`29ebec86a370fc38ebc8a34a5382ebd1551039ae`

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.

[1] https://github.com/bitgetwallet/bgw7702/tree/main/src

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

- Recommendation: 2
- Note: 2

ID	Severity	Description	Category	Status
1	-	Update `owner` when invoking the function `setSafe()`	Recommendation	Confirmed
2	-	Add an empty check in function `removeSigners()`	Recommendation	Confirmed
3	-	Potential centralization risks	Note	-
4	-	`EntryPoint` trust assumption and EIP-4337 compatibility	Note	-

The details are provided in the following sections.

2.1 Recommendation

2.1.1 Update `owner` when invoking the function `setSafe()`

Status Confirmed

Introduced by `Version 1`

Description In the `BGW7702Base` contract, the constructor sets both the `owner` and the `safe` address to the same `_safe` parameter, establishing them as a unified governance identity. However, the function `setSafe()` only updates the `safe` variable without invoking the function `transferOwnership()` to update the `owner`. After a function `setSafe()` invocation, the `owner` (which controls access to `onlyOwner` functions including `addSigners()`, `removeSigners()`, `addOperators()`, `removeOperators()`, and `setSafe()` itself) becomes stale relative to the intended controlling address, creating a persistent mismatch between the governance authority and the intended safe wallet.

```
59    function setSafe(address _safe) external onlyOwner {
60        _setSafe(_safe);
61    }
```

Listing 2.1: src/BGW7702Base.sol

```
115   function _setSafe(address _newSafe) internal {
116       _checkZero(_newSafe);
117       if (_newSafe == safe) revert SameAddress();
118       safe = _newSafe;
119       emit ChangeSafe(_newSafe);
120   }
```

Listing 2.2: src/BGW7702Base.sol

Suggestion Revise the logic to ensure that when the `safe wallet` is updated, the `owner` is updated accordingly in a timely manner.

Feedback from the project The function `setSafe()` is intended to update the recipient address for emergency withdrawals and does not involve any change of ownership. To change the owner, the function `transferOwnership()` must be called. The Safe wallet address is initially used as both the owner address and the emergency withdrawal recipient address; however, these two addresses may differ over time.

2.1.2 Add an empty check in function `removeSigners()`

Status Confirmed

Introduced by `Version 1`

Description In the `BGW7702Admin` contract, the `removeSigners()` function removes `signer` addresses from the `signer` set via function `_removeAddressSet()`. The comment on function `removeSigners()` states it should revert if the set would become empty, but the implementation does not enforce this invariant. The `owner` can remove all `signers` through one or more calls, leaving the `signer` set empty. Once empty, function `_checkSigner()` in function `verifySignatures()` will always fail because `ECDSA.recover` will return an address that cannot exist in an empty set, causing all function `aggregate()` and `aggregateV2()` calls to become permanently inoperable until new signers are re-added.

```
225   function removeSigners(address[] memory _signers) external onlyOwner {
226       _removeAddressSet(_signerSet, _signers);
227       emit RemoveSigners(_signers);
228   }
```

Listing 2.3: src/BGW7702Admin.sol

```
159   function _removeAddressSet(
160       EnumerableSet.AddressSet storage set,
161       address[] memory addrs
162   ) internal {
163       if (addrs.length == 0) revert EmptyArray();
164       bool success;
165       for (uint256 i = 0; i < addrs.length; i++) {
166           success = set.remove(addrs[i]);
167           if(!success) revert FailedRemove(addrs[i]);
168       }
169   }
```

Listing 2.4: src/BGW7702Base.sol

Suggestion Revise the logic to ensure that at least one `signer` remains after `signer` removal.

Feedback from the project No modification is required. By design, the `signer` set is permitted to be empty, so that if a `signer`'s private key is compromised and cannot be removed immediately, all existing `signer`s can first be removed and the intended new `signer`s can then be added.

2.2 Note

2.2.1 Potential centralization risks

Introduced by Version 1

Description In this project, several privileged roles (the owner/safe wallet, operator, and signer) can perform sensitive operations, introducing centralization risk. Specifically, the owner is able to reconfigure governance and authorization, including adding or removing signers and operators and updating the safe address. The operator is able to pause or unpause the protocol and perform asset rescue. If any privileged key is lost, compromised, or maliciously used, the protocol may face governance takeover, service disruption, or asset-handling risk at the admin layer.

Feedback from the project The owner and safe addresses will be designated as multi-signature addresses to mitigate the risk of private key compromise. And the operator role will be retained by the project team to ensure a rapid response in emergency situations.

2.2.2 EntryPoint trust assumption and EIP-4337 compatibility

Introduced by Version 1

Description The protocol integrates with an external EntryPoint contract for ERC-4337 account abstraction, including validateUserOp invocation, prefund handling, and execution settlement. The EntryPoint implementation is not included in this audit scope. This audit assumes that the deployed EntryPoint is correct, secure, and behaviorally compatible with this protocol's account logic, including validation semantics, gas accounting, and post-validation execution flow.

Feedback from the project The EntryPoint will use the official EntryPoint v0.8 address (stable release, widely adopted).


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS