BLOCKSEC

Security Audit
Report for solver

Date: August 10, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	BitgetWallet
Target	solver

Version History

Version	Date	Description
1.0	August 10, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-08-10 09:14:36+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 solver of BitgetWallet.

This project implements a Fusion-style resolver for settling 1inch Limit Order Protocol orders. During settlement, the resolver receives the maker asset, swaps it through an external aggregator into the taker asset, and allows the Limit Order Protocol to complete payment to the maker or receiver. The contract also includes executor access control, pause controls, token sweeping to configured vaults, and native ETH handling for WETH unwrap flows. Its safety model depends on trusted executors, trusted protocol dependencies, and off-chain validation of swap routes and profitability.

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
solver	`Version 1`	`b6e7b0b7baf8ab0a11f1e1eeace4894081892b06`

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/solver

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	-	Limit token approvals to required amounts	Recommendation	Confirmed
2	-	Reject the resolver address during vault configuration	Recommendation	Confirmed
3	-	Reliance on trusted off-chain logic	Note	-
4	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Recommendation

2.1.1 Limit token approvals to required amounts

Status Confirmed

Introduced by `Version 1`

Description The function `takerInteraction()` is invoked by the 1inch Limit Order Protocol during order settlement. When tokenIn is an ERC20 token, the resolver authorizes the `AGG` contract to spend `tokenIn` before invoking function `swap()` and subsequently authorizes the `LOPV4` contract to transfer `tokenOut`.

These approvals are managed through function `safeApproveMaxIfNeeded()`, which grants an allowance of `type(uint256).max` whenever the existing allowance is lower than the required amount. As a result, the approved contracts retain permission to transfer tokens beyond the amount required for the current settlement, unnecessarily increasing the resolver's exposure if either approved contract is compromised or behaves unexpectedly.

```
62
63   function safeApproveWithRetry(
64       address token,
65       address spender,
66       uint256 value
```

Listing 2.1: src/libs/TokenHelper.sol

Suggestion Use function `safeApproveWithRetry()` to approve only the amount required for each settlement operation and clear any residual allowance when it is no longer needed.

Feedback from the project The project stated that this resolver contract is primarily designed to fill limit orders on Ethereum mainnet and is therefore highly gas-sensitive. Since both the 1inch Router and the aggregator are trusted, non-upgradable third-party contracts that can transfer tokens only through caller-authorized interactions, granting them maximum allowances is acceptable and poses no security risk.

2.1.2 Reject the resolver address during vault configuration

Status Confirmed

Introduced by Version 1

Description The `FusionResolver` contract transfers accumulated token balances to configured vaults through function `claimTokens()`. Vault addresses are established through the constructor and functions `addVault()` and `setVault()`, which reject the zero address but permit the resolver's own address. If a vault is configured as `address(this)`, function `claimTokens()` transfers the relevant tokens back to the resolver and emits the `TokenClaimed` event even though no assets leave the contract. This configuration prevents the intended collection of accumulated proceeds and produces a misleading indication that the tokens were successfully claimed.

```
164
165   function addVault(address vault) external onlyOwner {
166       _checkZeroAddress(vault);
167       _vaults.push(vault);
168       emit VaultAdded(_vaults.length - 1, vault);
169   }
170
171   function setVault(uint256 index, address newVault) external onlyOwner {
172       _checkZeroAddress(newVault);
173       if (index >= _vaults.length) revert InvalidVaultIndex();
174       address oldVault = _vaults[index];
175       if (newVault == oldVault) revert SameValue();
176       _vaults[index] = newVault;
177       emit VaultUpdated(index, oldVault, newVault);
```

Listing 2.2: src/FusionResolver.sol

Suggestion Add a check that rejects `address(this)` in the vault configuration paths, consistent with the validation applied by function `setWithdrawer()`.

Feedback from the project The project stated that the owner may intentionally set a vault address to the resolver itself to disable that vault. In this case, the corresponding token remains in the resolver, while transfers to other valid vaults continue normally.

2.2 Note

2.2.1 Reliance on trusted off-chain logic

Introduced by Version 1

Description The protocol relies on trusted off-chain infrastructure to construct settlements and evaluate their profitability. An authorized executor prepares the interaction data embedded in the 1inch calldata, including the swap parameters and actions submitted to the external aggregator, and determines whether the expected settlement outcome is economically acceptable. The resolver validates certain execution conditions on-chain but does not enforce constraints governing route selection or profitability. Therefore, the correctness and economic

safety of each settlement depend on the executor system validating the generated calldata and expected outcome before transaction submission.

2.2.2 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., the `owner`, `operators`, `executors`, and `withdrawer`) can conduct sensitive operations, which introduces potential centralization risks. For example, the owner can manage privileged roles and vault addresses. Executors can submit order settlement data. Operators can pause or unpause order settlement. The withdrawer can transfer retained assets to the configured vaults. If the private keys of these privileged accounts are lost or maliciously exploited, the project could face significant risks.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS