BLOCKSEC

# Security Audit
# Report for Keeper Contract

**Date:** February 11, 2026 **Version:** 1.0

**Contact:** contact@blocksec.com

# Contents

**Chapter 1 Introduction** **1**
1.1 About the Audit Target . . . 1
1.2 Disclaimer . . . 2
1.3 Procedure of Auditing . . . 2
1.3.1 Security Issues . . . 2
1.3.2 Additional Recommendation . . . 3
1.4 Security Model . . . 3

**Chapter 2 Findings** **5**
2.1 Security Issue . . . 5
2.1.1 Improper construction of the trade hash . . . 5
2.1.2 Inconsistent access control design in the function `resolveDispute()` . . . 6
2.2 Recommendation . . . 6
2.2.1 Add non-zero value checks . . . 6
2.3 Note . . . 7
2.3.1 Potential centralization risks . . . 7

## Report Manifest

| Item | Description |
| --- | --- |
| Client | p2p-infra |
| Target | Keeper Contract |

## Version History

| Version | Date | Description |
| --- | --- | --- |
| 1.0 | February 11, 2026 | First release |

## Signature



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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-11-10 09:36:45+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 Keeper Contract of p2p-infra.

The Keeper Contract implements a smart contract–based conditional asset locking mechanism designed to facilitate peer-to-peer trades. The mechanism supports both ERC20 tokens and native blockchain assets.

Under this design, users initiate trade commitments by invoking the designated asset-locking function (escrow()), through which digital assets are transferred directly into the smart contract. The contract programmatically verifies signed trade instructions against a preconfigured authorized signer before locking assets according to predefined trade parameters.

The smart contract operates in a non-custodial manner. Assets are held and governed exclusively by on-chain logic and predefined execution conditions. Neither the platform operator nor any off-chain party has unilateral control over user assets outside the contract's encoded rules.

A relayer role is defined solely to trigger specific contract functions in accordance with predefined conditions, including releasing locked assets, cancelling pending commitments, or invoking dispute resolution pathways. The relayer does not have discretionary authority over assets; it can only execute functions permitted by the smart contract logic.

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

- src/Keeper.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 |
|---|---|---|
| Keeper Contract | `Version 1` | `e3010c5726710caa10b5bd02aa4b712238b3b084` |
| | `Version 2` | `718a978098d9404f1a80474ca2433619aab7075f` |

[1] https://github.com/p2p-infra/p2p-contracts

## 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.

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:

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

- **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.

# Chapter 2 Findings

In total, we found **two** potential security issues. Besides, we have **one** recommendation and **one** note.

- High Risk: 1
- Medium Risk: 1
- Recommendation: 1
- Note: 1

| ID | Severity | Description | Category | Status |
|---|---|---|---|---|
| 1 | High | Improper construction of the trade hash | Security Issue | Fixed |
| 2 | Medium | Inconsistent access control design in the function `resolveDispute()` | Security Issue | Fixed |
| 3 | - | Add non-zero value checks | Recommendation | Fixed |
| 4 | - | Potential centralization risks | Note | - |

The details are provided in the following sections.

## 2.1 Security Issue

### 2.1.1 Improper construction of the trade hash

**Severity** High

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the contract `Keeper`, the function `escrow()` allows users to create escrow orders with valid signatures. This function constructs a trade hash based on the provided inputs (e.g., `tradeId`) and transfers the corresponding amount of tokens from `msg.sender`. However, the construction of the trade hash is improper, as it lacks the variable `token` and a domain separator (i.e., `chainId` and `address(this)`). As a result, this may lead to potential DoS and replay attacks.

1. The lack of the variable `token` in the trade hash could lead to potential DoS issues. Specifically, an attacker can front-run the invocation of the function `escrow()` to create escrow orders with fake tokens, thereby preventing sellers from creating orders.

2. The lack of a domain separator in the trade hash makes used signatures replayable. As a result, a malicious user can create escrow orders in other Keeper contracts across different networks.

```
47   function escrow(
48       bytes32 tradeId,
49       address token,
50       address seller,
51       address buyer,
52       uint128 amount,
53       uint128 sellerFee,
54       uint128 sellerGasFee,
55       uint128 buyerFee,
```

```
56      uint128 buyerGasFee,
57      bytes calldata signature
58  ) external payable nonReentrant {
59      if (seller == address(0) || buyer == address(0)) revert InvalidAddress();
60      if (seller == buyer) revert InvalidAddress();
61      if (amount <= sellerFee + buyerFee + sellerGasFee + buyerGasFee) revert InvalidAmount();
62      if (escrows[tradeId].status != EscrowStatus.NONE) revert TradeAlreadyExists();
63
64      bytes32 hash = _getTradeHash(tradeId, seller, buyer, amount, sellerFee, sellerGasFee,
            buyerFee, buyerGasFee);
65      _verifySignature(hash, signature, signer);
66
67      if (token == ETH_ADDRESS) {
68          if (msg.value != amount) revert InvalidValue();
69      } else {
70          if (msg.value != 0) revert InvalidValue();
71          IERC20(token).safeTransferFrom(msg.sender, address(this), amount);
72      }
```

**Listing 2.1:** src/Keeper.sol

**Impact** Potential DoS and replay attacks.

**Suggestion** Revise the construction of the trade hash.

### 2.1.2 Inconsistent access control design in the function `resolveDispute()`

**Severity** Medium

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the contract `Keeper`, the function `resolveDispute()` allows the relayer role to resolve disputes between sellers and buyers. However, according to the `README.md` file, this function should be accessible only to the arbitrator role. As a result, the inconsistent implementation of access control may lead to incorrect functionality.

```
97  function resolveDispute(bytes32 tradeId, uint8 buyerPercent) external nonReentrant onlyRole(
        RELAYER_ROLE) {
```

**Listing 2.2:** src/Keeper.sol

**Impact** The inconsistent implementation of access control may lead to incorrect functionality.

**Suggestion** Revise the inconsistent implementation in the function `resolveDispute()`.

## 2.2 Recommendation

### 2.2.1 Add non-zero value checks

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the functions `constructor()`, `setSigner()` and `setRecipient()`, several inputs (e.g., `_recipient`) are not checked to ensure they are not zero. It is recommended to add such checks to prevent potential misoperations or loss of funds.

```
25   constructor(address _signer, address _recipient) {
26       _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
27       signer = _signer;
28       recipient = _recipient;
```

**Listing 2.3:** src/Keeper.sol

```
31   function setSigner(address _signer) external onlyRole(DEFAULT_ADMIN_ROLE) {
32       address old = signer;
33       signer = _signer;
```

**Listing 2.4:** src/Keeper.sol

```
37   function setRecipient(address _recipient) external onlyRole(DEFAULT_ADMIN_ROLE) {
38       address old = recipient;
39       recipient = _recipient;
```

**Listing 2.5:** src/Keeper.sol

**Suggestion** Add non-zero value checks accordingly.

## 2.3 Note

### 2.3.1 Potential centralization risks

**Introduced by** `Version 1`

**Description** In this project, several privileged roles (e.g., the role `RELAYER_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, the role `RELAYER_ROLE` can release or cancel an escrow. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS