BLOCKSEC

Security Audit

Report for EIP3009Upgradeable Contract

Date: July 9, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Bitget
Target	EIP3009Upgradeable Contract

Version History

Version	Date	Description
1.0	July 9, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-07-09 06:43:34+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 EIP3009Upgradeable Contract of Bitget.

The EIP3009Upgradeable Contract extends `RWAToken` with `EIP3009Upgradeable`, enabling users to authorize transfers off-chain and have them submitted on-chain by relayers. `RWAToken` remains a compliance-aware rebase ERC-20 token, with all asset movement ultimately reusing the existing pause, freeze, sanctions oracle, and rebase accounting checks. This extension adds `transferWithAuthorization()`, `receiveWithAuthorization()`, and `cancelAuthorization()` entry points, along with EIP-712 signature verification, nonce-based replay protection, time-window validation, and support for both EOA and ERC-1271 signatures. Relayers only submit user authorizations and do not directly hold user assets; whether an authorization can be executed still depends on the current on-chain state, including pause status, compliance checks, account restrictions, and the rebase index.

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.

- src/EIP3009Upgradeable.sol
- src/RWAToken.sol
- src/interfaces/IRWAToken.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
EIP3009Upgradeable Contract	`Version 0`	`95ec4e045b38b24ea61cac9de0927a68834f1ba1`
	`Version 1`	`f37d8fedb520a4bf82fa4dc37940a2a098e4c54c`

[1] https://github.com/blcokchainbg/rwa-evm

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:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.

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

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

- **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 **one** potential security issue. Besides, we have **one** recommendation and **two** notes.

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

ID	Severity	Description	Category	Status
1	Low	Lack of `rebaseIndex` binding in signed authorizations	Security Issue	Confirmed
2	-	Add explicit interfaces to invoke crucial functions	Recommendation	Confirmed
3	-	Integrator assumption	Note	-
4	-	Authorization cancellation can be front-run	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Lack of `rebaseIndex` binding in signed authorizations

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the contract `EIP3009Upgradeable`, the functions `_receiveWithAuthorization()` and `_transferWithAuthorization()` transfer tokens based on a signed authorization. However, the authorization binds only the transfer amount (i.e., `value`). It does not bind the expected `rebaseIndex`. At execution time, the contract converts `value` into raw balances using the current `rebaseIndex`. If a rebase occurs between signing and execution, the same `value` may no longer correspond to the signer's originally intended raw amount and may lead to a revert if a reverse split leaves the signer with insufficient balance.

```
36        "TransferWithAuthorization(address from,address to,uint256 value,uint256 validAfter,uint256
              validBefore,bytes32 nonce)"
37    );
38
39    bytes32 public constant RECEIVE_WITH_AUTHORIZATION_TYPEHASH = keccak256(
40        "ReceiveWithAuthorization(address from,address to,uint256 value,uint256 validAfter,uint256
              validBefore,bytes32 nonce)"
41    );
```

Listing 2.1: src/EIP3009Upgradeable.sol

```
188   function _transfer(
189       address from,
```

```
190         address to,
191         uint256 amount
192     ) internal virtual {
193         if (from == address(0)) revert ZeroAddress();
194         if (to == address(0)) revert ZeroAddress();
195
196         uint256 bal = balanceOf(from);
197         if (bal < amount) revert InsufficientBalance(bal, amount);
198
199         RebaseStorage storage $ = _getRebaseStorage();
200         uint256 rawAmount = amount * $._rebaseIndex;
201         $._rawBalances[from] -= rawAmount;
202         $._rawBalances[to] += rawAmount;
203
204         emit Transfer(from, to, amount);
205     }
```

Listing 2.2: src/RWATokenRebaseUpgradeable.sol

```
144     /// @dev Core transfer flow shared by the `(v,r,s)` and `bytes signature` overloads.
145     function _transferWithAuthorization(
146         address from,
147         address to,
148         uint256 value,
149         uint256 validAfter,
150         uint256 validBefore,
151         bytes32 nonce,
152         bytes memory signature
153     ) internal virtual {
154         _beforeAuthorizationUse();
155         _requireValidAuthorization(from, nonce, validAfter, validBefore);
156         bytes32 structHash = keccak256(
157             abi.encode(TRANSFER_WITH_AUTHORIZATION_TYPEHASH, from, to, value, validAfter,
                    validBefore, nonce)
158         );
159         _verifyAndConsumeAuthorization(from, nonce, structHash, signature);
160         _transferWithChecks(from, to, value);
161     }
162
163     /// @dev Core receive flow; requires the payee (`to`) to be the caller.
164     function _receiveWithAuthorization(
165         address from,
166         address to,
167         uint256 value,
168         uint256 validAfter,
169         uint256 validBefore,
170         bytes32 nonce,
171         bytes memory signature
172     ) internal virtual {
173         _beforeAuthorizationUse();
174         if (to != msg.sender) revert CallerMustBePayee();
175         _requireValidAuthorization(from, nonce, validAfter, validBefore);
176         bytes32 structHash =
```

```
177             keccak256(abi.encode(RECEIVE_WITH_AUTHORIZATION_TYPEHASH, from, to, value, validAfter,
                    validBefore, nonce));
178         _verifyAndConsumeAuthorization(from, nonce, structHash, signature);
179         _transferWithChecks(from, to, value);
180     }
```

Listing 2.3: src/EIP3009Upgradeable.sol

Impact The transferred raw balance may differ from the signer's original expectation.

Suggestion Include the expected `rebaseIndex` in the signed authorization, and reject the authorization when the current index differs.

Feedback from the project The project states that this behavior is by design. The `rebaseIndex` is updated infrequently. Additionally, the documentation advises integrators to configure a short valid time for the authorization, and the relayer checks the sender's balance before submitting a transaction.

2.2 Recommendation

2.2.1 Add explicit interfaces to invoke crucial functions

Status Confirmed

Introduced by `Version 1`

Description In the contract `RWAToken`, the EIP-3009 authorization functions `transferWithAuthorization()`, `receiveWithAuthorization()`, `cancelAuthorization()`, and `authorizationState()` are externally callable via the inherited contract `EIP3009Upgradeable`. However, the interface `IRWAToken` does not declare them, resulting in an unclear API surface for integrators.

Impact The incomplete interface leads to an unclear API surface for integrators.

Suggestion Add interfaces for crucial functionality in the `IRWAToken`.

Feedback from the project The project states that these interfaces and the EIP-3009 support scope are clarified in the integration document.

2.3 Note

2.3.1 Integrator assumption

Introduced by `Version 1`

Description Since contract `RWAToken` supports EIP-3009, the integrators should be aware of the following considerations.

First, when the transfer is initiated from another contract, integrators should use function `receiveWithAuthorization()` instead of `transferWithAuthorization()`. Since function `transferWithAuthorization()` does not check `msg.sender`, anyone can submit the signature and front-run it, causing unexpected behavior.

Second, when multiple authorizations are submitted at once, their execution order is decided by relayers and miners, not the sender. If authorizations are to be executed sequentially,

the integrator should submit them one at a time and wait for confirmation before submitting the next one.

Feedback from the project The project states that these assumptions are added to the integration document.

2.3.2 Authorization cancellation can be front-run

Introduced by `Version 1`

Description The function `cancelAuthorization()` cancels a signed transfer authorization by consuming the same nonce, so whichever transaction is mined first consumes the nonce. If the original signed authorization has already been shared with a relayer or exposed to the mempool, the `cancelAuthorization()` execution can be front-run by executing the transfer first, causing the cancellation to fail.

Feedback from the project The project states that this is an inherent limitation of all authorization schemes based on one-time nonces, including the standard EIP-3009, and the integration document added this limitation.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS