BLOCKSEC

Security Audit
Report for rwa-evm

Date: July 31, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Bitget
Target	rwa-evm

Version History

Version	Date	Description
1.0	July 31, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-07-31 09:10:41+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 rwa-evm of Bitget.

The rwa-evm project refines token initialization and corporate action handling. Newly created RWA tokens can now be launched with a caller-specified starting rebase index rather than a fixed default, allowing a token to open already aligned with a share ratio inherited from off-chain records or an earlier deployment. The starting index is bounded and recorded on creation. Rebases are restricted to when the token is paused, so transfers cannot interleave with an index update. A factory may also be deployed against an existing upgrade beacon, so that tokens it creates share the implementation and upgrade authority of earlier token deployments.

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/interfaces/IRWAToken.sol
- src/interfaces/IRWATokenFactory.sol
- src/RWAToken.sol
- src/RWATokenFactory.sol
- src/RWATokenRebaseUpgradeable.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
rwa-evm	`Version 0`	`f37d8fedb520a4bf82fa4dc37940a2a098e4c54c`
	`Version 1`	`d607918fe884eba4fa7a54d194acb23f071c58ba`

[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 have **one** recommendation and **three** notes.

- Recommendation: 1
- Note: 3

ID	Severity	Description	Category	Status
1	-	Align the initial index comment with implementation	Recommendation	Confirmed
2	-	Asymmetric pause requirements for rebase execution	Note	-
3	-	Potential centralization risks	Note	-
4	-	Outdated `RWATokenFactory` deployment helper	Note	-

The details are provided in the following sections.

2.1 Recommendation

2.1.1 Align the initial index comment with implementation

Status Confirmed

Introduced by `Version 1`

Description In the contract `RWATokenRebaseUpgradeable`, the comment above function `__RWATokenRebase_init()` states that the default initial index is `1e18`. However, the function does not apply a default value and directly assigns the supplied `startIndex_` to `_rebaseIndex` after validating its range. It is recommended to align the comment with the implementation.

```
56   /// Default INITIAL_INDEX=1e18;
57   function __RWATokenRebase_init(
58       string memory name_,
59       string memory symbol_,
60       uint256 startIndex_
61   ) internal onlyInitializing {
62       if (startIndex_ == 0 || startIndex_ > MAX_INDEX) revert InvalidIndex();
63
64       RebaseStorage storage $ = _getRebaseStorage();
65       $._name = name_;
66       $._symbol = symbol_;
67       $._rebaseIndex = startIndex_;
68   }
```

Listing 2.1: src/RWATokenRebaseUpgradeable.sol

Suggestion Revise the comment to state that `startIndex_` must be explicitly supplied within the supported range.

Feedback from the project The project has decided to retain the existing comment without modification. The default initial index is conventionally 1e18, but `startIndex_` must be explicitly

calculated and provided by the off-chain system. The contract only validates that it falls within `(0, MAX_INDEX]` and intentionally avoids silent defaults, ensuring missing or invalid inputs revert immediately during token creation.

2.2 Note

2.2.1 Asymmetric pause requirements for rebase execution

Introduced by `Version 1`

Description In the contract `RWAToken`, function `rebase()` is expected to execute an approved rebase action while token transfers are halted. The function now requires the token to be paused, but it invokes function `validateAndExecuteAction()` in contract `CorporateActionOracle`, which requires the oracle to be unpaused. This creates an operational requirement where the token must be paused and the oracle must remain unpaused during rebase execution.

```
176   function rebase(
177       bytes32 id,
178       uint32 from,
179       uint32 to
180   ) external virtual onlyRole(CONFIGURER_ROLE) whenPaused {
181       _checkNotDelisted();
182
183       bytes memory payload = abi.encode(from, to);
184       ICorporateActionOracle.ActionType actionType = ICorporateActionOracle.ActionType.Rebase;
185       bytes32 dataHash = _computeDataHash(actionType, payload);
186
187       RWATokenStorage storage $ = _getRWATokenStorage();
188       $.actionOracle.validateAndExecuteAction(id, dataHash);
189
190       _rebaseBySplit(from, to);
191   }
```

Listing 2.2: src/RWAToken.sol

```
89   function validateAndExecuteAction(
90       bytes32 id,
91       bytes32 dataHash
92   ) external whenNotPaused {
```

Listing 2.3: src/CorporateActionOracle.sol

Feedback from the project The project confirmed that this is an intended behavior, as the token must be paused during a rebase to ensure an atomic index update while the oracle must remain active, with the required sequence documented in the operational SOP.

2.2.2 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., the factory deployer, `DEFAULT_ADMIN_ROLE`, and `CONFIGURER_ROLE`) can conduct sensitive operations, which introduces potential centralization risks. For example, privileged deployment and configuration paths can select an `existingBeacon_` for `RWATokenFactory`, determine the upgrade authority and implementation used by newly created token proxies, and provide the `startIndex_` that seeds each new token's initial `rebaseIndex`. If an unintended beacon is configured, newly created tokens may fail to deploy or may be placed under an unexpected upgrade authority. If an incorrect `startIndex_` is provided, the token may start from an accounting scale that downstream systems cannot reconcile with the intended corporate-action history. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

Feedback from the project The project confirmed that privileged roles are required for regulated RWA operations and mitigated through multisignature governance, deployment checks, and secure key management.

2.2.3 Outdated `RWATokenFactory` deployment helper

Introduced by `Version 1`

Description In the deployment helper, function `_deployRWATokenFactory()` is expected to construct the deployment bytecode for contract `RWATokenFactory` with all required constructor dependencies. However, the constructor now accepts eight arguments, including the optional migration beacon `existingBeacon_`, while the helper manually appends only the previous seven ABI-encoded arguments to `type(RWATokenFactory).creationCode`. Deployments that rely on this helper must encode `existingBeacon_`, using the zero address when no existing beacon is required.

```
23    constructor(
24        address defaultAdmin_,
25        address vault_,
26        address operator_,
27        address configurer_,
28        address tokenManager_,
29        address sanctionsList_,
30        address actionOracle_,
31        address existingBeacon_
32    ) RWACommon(defaultAdmin_, vault_, operator_, configurer_) {
33        _zeroAddressCheck(tokenManager_);
34        _zeroAddressCheck(sanctionsList_);
35        _zeroAddressCheck(actionOracle_);
36
37        admin = defaultAdmin_;
38        vault = vault_;
39        operator = operator_;
40        configurer = configurer_;
41
42        if (existingBeacon_ == address(0)) {
43            address tokenImplementation = address(new RWAToken());
44            defaultBeacon = new UpgradeableBeacon(tokenImplementation, admin);
45        } else {
```

```
46          defaultBeacon = UpgradeableBeacon(existingBeacon_);
47      }
48
49      tokenManager = IRWATokenManager(tokenManager_);
50      sanctionsList = sanctionsList_;
51      actionOracle = actionOracle_;
52  }
```

Listing 2.4: src/RWATokenFactory.sol

```
197 function _deployRWATokenFactory(
198     address tokenManager,
199     address sanctionsList,
200     address corporateActionOracle
201 ) internal returns (address) {
202     bytes memory bytecode = abi.encodePacked(
203         type(RWATokenFactory).creationCode,
204         abi.encode(
205             deployer, //ADMIN
206             vm.envAddress("VAULT"),
207             vm.envAddress("OPERATOR"),
208             vm.envAddress("CONFIGURER"),
209             tokenManager,
210             sanctionsList,
211             corporateActionOracle
212         )
213     );
214     address addr = factory.deploy(bytecode, _getSalt("rwaTokenFactory"));
215     return addr;
216 }
```

Listing 2.5: script/RWADeploy.s.sol

Feedback from the project The project confirmed that `script/RWADeploy.s.sol` is intended only for full deployments on new chains, where `existingBeacon_` is always `address(0)` by design. Migration and cross-chain scenarios will use a separate deployment script with strict input validation.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS