BLOCKSEC

Security Audit Report for mStocks Protocol

Date: July 20, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Mantle
Target	mStocks Protocol

Version History

Version	Date	Description
1.0	July 20, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-07-27 02:51:06+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 mStocks Protocol of Mantle.

The project is a real-world-asset (RWA) tokenization platform deployed on Mantle Network that mints RWA tokens (i.e., `ScaledToken`) representing tokenized stocks and equities. It adopts ERC-8056, which separates a token's raw on-chain amount from its UI representation via a configurable multiplier, enabling efficient handling of stock splits without actual token minting or transfers. The protocol enforces an on-chain compliance layer with per-token blocklists and a global sanctions list. A two-tier pause mechanism operates at both the token-specific and protocol-wide levels. Every token transfer is subject to both controls. Additionally, the project adopts a role-based, separation-of-duties access control model, in which `DEFAULT_ADMIN_ROLE` may only grant and revoke roles, while distinct operational roles govern minting/burning, multiplier scheduling, compliance, and pausing.

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
mStocks-protocol	`Version 1`	`f09a8e4979afa603bab62b8fda83a8ff52a405ca`
	`Version 2`	`eed6c34dfa90a6488aa8b4006c233c4c34e04608`

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,

[1] https://github.com/mantle-xyz/mStocks-protocol

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.
- **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 found **seven** potential security issues. Besides, we have **seven** recommendations and **seven** notes.

- Low Risk: 7
- Recommendation: 7
- Note: 7

ID	Severity	Description	Category	Status
1	Low	Potential circumvention of role isolation rule	Security Issue	Confirmed
2	Low	Lack of upgrade capability in contract `TokenIssuer`	Security Issue	Fixed
3	Low	Circumvention of contract-only privilege restrictions	Security Issue	Confirmed
4	Low	Incorrect pause logic on function `withdraw()`	Security Issue	Partially Fixed
5	Low	Lack of token validation in contract `TokenIssuer`	Security Issue	Confirmed
6	Low	Lack of role revocation for a publicly derivable wallet	Security Issue	Fixed
7	Low	Potential overwriting of a pending multiplier	Security Issue	Confirmed
8	-	Emit issuer transfer events	Recommendation	Fixed
9	-	Remove the redundant codes	Recommendation	Fixed
10	-	Align annotations with implementation	Recommendation	Fixed
11	-	Add validations on the symbol	Recommendation	Fixed
12	-	Avoid overflow in function `getTokens()`	Recommendation	Fixed
13	-	Add a length check for `newList`	Recommendation	Fixed
14	-	Add a uniqueness check on the `assetCode`	Recommendation	Fixed
15	-	Potential centralization risks	Note	-
16	-	Ensure migration of crucial configurations for existing tokens	Note	-
17	-	Dust visibility due to precision loss	Note	-
18	-	Immutable EIP-712 domain	Note	-
19	-	Token lifecycle management	Note	-
20	-	Compliance model	Note	-
21	-	Allowances are denominated in raw units	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Potential circumvention of role isolation rule

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The project is designed to enforce a strict separation of duties in its role model. However, this restriction can be circumvented. Specifically, function `grantRole()` only prevents a `DEFAULT_ADMIN_ROLE` holder from granting a role to itself. It does not prevent an operational role from being assigned to another address that already holds `DEFAULT_ADMIN_ROLE`, or vice versa. The same issue exists across multiple contracts, including `ComplianceRegistry`, `PauseRegistry`, `ScaledToken`, and `TokenIssuer`.

Additionally, the function `__scaledToken_init()` in contract `ScaledToken` does not check the input arrays `issuers_` and `multiplierSchedulers_`. If the `admin_` address is included in these arrays, the `admin_` can be granted `ISSUER_ROLE` or `MULTIPLIER_SCHEDULER_ROLE`, violating the role isolation rule.

```
30## Role model -strict separation of duties
```

Listing 2.1: README.md

```
21///       Self-grant is disallowed so the admin and operator must be distinct accounts.
```

Listing 2.2: src/tokenIssuer/TokenIssuer.sol

```
108       _grantRole(DEFAULT_ADMIN_ROLE, admin_);
109       _setRoleAdmin(ISSUER_ROLE, DEFAULT_ADMIN_ROLE);
110       _setRoleAdmin(MULTIPLIER_SCHEDULER_ROLE, DEFAULT_ADMIN_ROLE);
111       for (uint256 i = 0; i < issuers_.length; ++i) {
112           if (issuers_[i] == address(0)) revert ZeroAddress();
113           _grantRole(ISSUER_ROLE, issuers_[i]);
114       }
115       for (uint256 i = 0; i < multiplierSchedulers_.length; ++i) {
116           if (multiplierSchedulers_[i] == address(0)) revert ZeroAddress();
117           _grantRole(MULTIPLIER_SCHEDULER_ROLE, multiplierSchedulers_[i]);
118       }
```

Listing 2.3: src/scaledToken/ScaledToken.sol

```
233   function grantRole(bytes32 role, address account)
234       public
235       virtual
236       override(AccessControlUpgradeable, IAccessControl)
237   {
238       if (account == _msgSender()) revert SelfGrantNotAllowed();
239       super.grantRole(role, account);
240   }
```

Listing 2.4: src/scaledToken/ScaledToken.sol

Impact One account can be granted `DEFAULT_ADMIN_ROLE` and operational roles, violating the role isolation rule.

Suggestion Add checks to prevent a role holder from being granted another role.

Feedback from the project The project confirmed this issue and decided not to apply code changes. Separation of duties will instead be enforced through multisig-controlled SOPs, non-overlapping role assignments, pre-grant checks, and event monitoring.

2.1.2 Lack of upgrade capability in contract `TokenIssuer`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The project documentation states that the contract is a UUPS-style contract deployed behind the `ERC1967Proxy` pattern and can be upgraded by an admin. The deploy script follows the same logic. However, contract `TokenIssuer` does not implement any upgrade function required by the UUPS pattern. As a result, any upgrade invocation reverts, and the implementation is permanently frozen at its initial version, breaking the intended upgradeable design.

```
22contract TokenIssuer is
23    Initializable,
24    AccessControlEnumerableUpgradeable,
25    ReentrancyGuard,
26    PausableUpgradeable
```

Listing 2.5: src/tokenIssuer/TokenIssuer.sol

```
161    function _deployTokenIssuer(Config memory cfg, bool broadcasterIsAdmin)
162        internal
163        returns (TokenIssuer issuer)
164    {
165        TokenIssuer impl = new TokenIssuer();
166        bytes memory initData = abi.encodeWithSelector(TokenIssuer.initialize.selector, cfg.admin);
167        issuer = TokenIssuer(payable(address(new ERC1967Proxy(address(impl), initData))));
```

Listing 2.6: script/Deploy.s.sol

Impact The contract `TokenIssuer` cannot be upgraded and is permanently frozen at its initial version.

Suggestion Inherit the contract `UUPSUpgradeable` and implement the function `_authorizeUpgrade()`.

2.1.3 Circumvention of contract-only privilege restrictions

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the contracts `ScaledToken` and `ScaledTokenFactory`, functions `_grantRole()`, `_assertCompliance()` and `_assertPauseManager()` restrict `ISSUER_ROLE`, `_compliance` and `_pauseManager` to contracts by checking the target account's code length. However, on chains that support EIP-7702, an EOA can authorize a delegation to a contract. This causes `account.code.length` and `EXTCODESIZE` to be non-zero while the account is still controlled by a single EOA key. Such an address can pass the role-assignment and configuration-time contract checks and be assigned as a contract-gated privileged actor.

```
243   ///     are seeded via internal _grantRole, bypassing the public override). Enforces
244   ///     that ISSUER_ROLE may only be granted to a contract.
245   function _grantRole(bytes32 role, address account)
246       internal
247       virtual
248       override(AccessControlEnumerableUpgradeable)
249       returns (bool)
250   {
251       if (role == ISSUER_ROLE && account.code.length == 0) revert IssuerNotContract();
252       return super._grantRole(role, account);
253   }
```

Listing 2.7: src/scaledToken/ScaledToken.sol

```
147   function _assertCompliance(address _compliance) internal view {
148       uint256 size;
149       assembly { size := extcodesize(_compliance) }
150       if (size == 0) revert InvalidComplianceContract();
151       try IComplianceRegistry(_compliance).checkIsCompliant(address(this), address(this)) {}
152       catch (bytes memory) {
153           revert InvalidComplianceContract();
154       }
155   }
```

Listing 2.8: src/scaledToken/ScaledTokenFactory.sol

Impact The intended invariant that `ISSUER_ROLE`, `_compliance`, and `_pauseManager` are contract accounts can be circumvented.

Suggestion Reject EIP-7702 delegation accounts for all contract-gated privileged actors.

Feedback from the project The project stated that the EIP-7702 risk is acknowledged and accepted. Code-length checks are only a defense-in-depth measure, while privileged targets are team-deployed contracts whose identities are verified before assignment.

2.1.4 Incorrect pause logic on function `withdraw()`

Severity Low

Status Partially Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `TokenIssuer`, function `withdraw()` performs an emergency sweep of the native or ERC-20 balance to the admin, and is gated by the modifier `whenNotPaused`. When the contract is paused, the admin must unpause it to sweep funds, which enables functions

`mint()`, `burn()`, and `transfer()` for every active `OPS_ROLE` holder. This creates an unnecessary exposure window during an emergency. Additionally, functions `addApprovedRecipient()` and `removeApprovedRecipient()` have a similar issue.

```
116   /// @notice Emergency sweep of native (`token == address(0)`) or ERC-20 balance to the caller.
117   function withdraw(address token) external whenNotPaused onlyRole(DEFAULT_ADMIN_ROLE)
          nonReentrant {
118       uint256 assetBalance;
119       address to = _msgSender();
120       if (token == address(0)) {
121           assetBalance = address(this).balance;
122           if (assetBalance != 0) {
123               (bool ok,) = payable(to).call{value: assetBalance}("");
124               require(ok, "TokenIssuer: native withdraw failed");
125           }
126       } else {
127           assetBalance = IERC20(token).balanceOf(address(this));
128           if (assetBalance != 0) {
129               IERC20(token).safeTransfer(to, assetBalance);
130           }
131       }
132       emit Withdrawn(token, to, assetBalance);
133   }
```

Listing 2.9: src/tokenIssuer/TokenIssuer.sol

```
 86   function addApprovedRecipient(address recipient) external whenNotPaused onlyRole(
          DEFAULT_ADMIN_ROLE) {
 87       if (recipient == address(0)) revert ZeroAddress();
 88       if (recipient == address(this)) revert CannotBeSelf();
 89       if (recipientIndex[recipient] != 0) revert RecipientAlreadyApproved();
 90       approvedRecipients.push(recipient);
 91       recipientIndex[recipient] = approvedRecipients.length;
 92       emit RecipientApproved(recipient);
 93   }
 94
 95   function removeApprovedRecipient(address recipient) external whenNotPaused onlyRole(
          DEFAULT_ADMIN_ROLE) {
 96       if (recipient == address(0)) revert ZeroAddress();
 97       uint256 idx = recipientIndex[recipient];
 98       if (idx == 0) revert RecipientNotApproved();
 99       if (idx != approvedRecipients.length) {
100           approvedRecipients[idx - 1] = approvedRecipients[approvedRecipients.length - 1];
101           recipientIndex[approvedRecipients[idx - 1]] = idx;
102       }
103       approvedRecipients.pop();
104       delete recipientIndex[recipient];
105       emit RecipientRevoked(recipient);
106   }
```

Listing 2.10: src/tokenIssuer/TokenIssuer.sol

Impact The admin must unpause the contract before sweeping funds, leading to an unnecessary exposure window during an emergency.

Suggestion Remove the modifier `whenNotPaused` from the functions `withdraw()`, `addApprovedRecipient()`, and `removeApprovedRecipient()`.

Feedback from the project The project confirmed that functions `addApprovedRecipient()` and `removeApprovedRecipient()` will retain the modifier `whenNotPaused`, as whitelist management is not an emergency action. If changes are required during a pause, operators can first revoke `OPS_ROLE`, thereby disabling their privileges, and then unpause and update the whitelist without exposing operational functions.

2.1.5 Lack of token validation in contract `TokenIssuer`

Severity Low

Status Confirmed

Introduced by `Version 1`

Description In the contract `TokenIssuer`, functions `mint()`, `burn()`, and `transfer()` are expected to operate on managed `ScaledToken` instances. However, each function accepts an arbitrary `token` address and does not verify that the `ScaledToken` instance was created by the protocol factory. As a result, an `OPS_ROLE` holder can route issuer operations through arbitrary token contracts, and function `transfer()` can move any ERC-20 balance held by the issuer to an approved recipient.

```
16/// @notice Custodial issuer that holds ISSUER_ROLE on one or more ScaledToken contracts
17///         and routes mint/burn flow through a whitelisted set of approved recipients.
18///
19///         Admin (DEFAULT_ADMIN_ROLE) manages the recipient whitelist, pauses the issuer,
20///         and may emergency-sweep assets. OPS_ROLE performs day-to-day mint / burn / transfer.
```

Listing 2.11: src/tokenIssuer/TokenIssuer.sol

```
169   function transfer(address token, address recipient, uint256 amount)
170       external
171       whenNotPaused
172       onlyRole(OPS_ROLE)
173       nonReentrant
174   {
175       if (!isApprovedRecipient(recipient)) revert RecipientNotApproved();
176       IERC20(token).safeTransfer(recipient, amount);
177   }
```

Listing 2.12: src/tokenIssuer/TokenIssuer.sol

```
136   ///         keeps the inventory on the issuer (recipient == self) or forwards to an approved
137   ///         recipient.
138   function mint(address token, address recipient, uint256 amount)
139       external
140       whenNotPaused
141       onlyRole(OPS_ROLE)
142       nonReentrant
143   {
```

```
144         if (recipient != address(this) && !isApprovedRecipient(recipient)) revert
                RecipientNotApproved();
145         if (!IMintAndBurnable(token).mint(amount)) revert MintFailed();
146         if (recipient != address(this)) {
147             IERC20(token).safeTransfer(recipient, amount);
148         }
149         emit Minted(address(this), token, recipient, amount);
150     }
```

Listing 2.13: src/tokenIssuer/TokenIssuer.sol

```
153     ///       recipient that has pre-approved the issuer to pull tokens.
154     function burn(address token, address from, uint256 amount)
155         external
156         whenNotPaused
157         onlyRole(OPS_ROLE)
158         nonReentrant
159     {
160         if (from != address(this) && !isApprovedRecipient(from)) revert FromNotApproved();
161         if (from != address(this)) {
162             IERC20(token).safeTransferFrom(from, address(this), amount);
163         }
164         if (!IMintAndBurnable(token).burn(amount)) revert BurnFailed();
165         emit Burned(address(this), token, from, amount);
166     }
167
168     /// @notice Transfers `amount` of `token` already held by this issuer to an approved recipient.
169     function transfer(address token, address recipient, uint256 amount)
170         external
171         whenNotPaused
172         onlyRole(OPS_ROLE)
173         nonReentrant
174     {
175         if (!isApprovedRecipient(recipient)) revert RecipientNotApproved();
176         IERC20(token).safeTransfer(recipient, amount);
177     }
```

Listing 2.14: src/tokenIssuer/TokenIssuer.sol

```
5///       Both operations act on the caller's balance.
6interface IMintAndBurnable {
7     function mint(uint256 amount) external returns (bool);
8     function burn(uint256 amount) external returns (bool);
9}
```

Listing 2.15: src/tokenIssuer/IMintAndBurnable.sol

Impact The issuer's operational role can operate on assets outside the intended managed token scope.

Suggestion Add an admin-controlled token allowlist or bind `TokenIssuer` to the factory registry, then require `mint()`, `burn()`, and operational `transfer()` to use allowed tokens only.

Feedback from the project The project confirmed this issue and decided not to apply code changes. The risk is mitigated through `OPS_ROLE` restrictions, approved-recipient controls, the

target token's `ISSUER_ROLE` requirement, managed-token-only inventory policies, admin recovery via function `withdraw()`, multisig-controlled operations, and event monitoring.

2.1.6 Lack of role revocation for a publicly derivable wallet

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the script contract `Smoke`, the annotation implies the script runs tests against an already-deployed token on a real chain. The function `_setupOps()` derives an operational wallet through `vm.createWallet("mstocks-smoke-ops")`, whose private key equals `keccak256("mstocks-smoke-ops")` and grants this derivable address operational roles. However, the function never revokes them after the script finishes executing. If the repository is made public, anyone can derive the private key of this address and perform privileged operations.

```
13/// @notice End-to-end smoke tests against a deployed ScaledToken on a real chain.
```

Listing 2.16: script/Smoke.s.sol

```
346   function _ops() internal returns (Vm.Wallet memory) {
347       string memory label = vm.envOr("SMOKE_OPS_LABEL", string("mstocks-smoke-ops"));
348       return vm.createWallet(label);
349   }
```

Listing 2.17: script/Smoke.s.sol

```
397   function _setupOps(ScaledToken token) internal returns (Vm.Wallet memory ops, TokenIssuer
          issuer) {
398       ops = _ops();
399       Vm.Wallet memory user = _user();
400       issuer = _getOrDeployIssuer();
401
402       ComplianceRegistry c = ComplianceRegistry(address(token.compliance()));
403       PauseRegistry p = PauseRegistry(address(token.pauseManager()));
404
405       bytes32 tIssuer = token.ISSUER_ROLE();
406       bytes32 tScheduler = token.MULTIPLIER_SCHEDULER_ROLE();
407       bytes32 issuerOps = issuer.OPS_ROLE();
408       bytes32 cOps = c.OPS_ROLE();
409       bytes32 pOps = p.OPS_ROLE();
410
411       bool needIssuerOnIssuer = !token.hasRole(tIssuer, address(issuer));
412       bool needOpsOnIssuer = !issuer.hasRole(issuerOps, ops.addr);
413       bool needApprovedRecipient = !issuer.isApprovedRecipient(user.addr);
414       bool needScheduler = !token.hasRole(tScheduler, ops.addr);
415       bool needCOps = !c.hasRole(cOps, ops.addr);
416       bool needPOps = !p.hasRole(pOps, ops.addr);
417       bool needGas = ops.addr.balance
418           < vm.envOr("SMOKE_USER_GAS_THRESHOLD", DEFAULT_GAS_THRESHOLD);
419
420       if (
```

```
421             !needIssuerOnIssuer && !needOpsOnIssuer && !needApprovedRecipient && !needScheduler
422                 && !needCOps && !needPOps && !needGas
423         ) {
424             return (ops, issuer);
425         }
426
427         console.log("  setup ops:         ", ops.addr);
428         vm.startBroadcast();
429         if (needGas) _topUpAddr(ops.addr);
430         if (needIssuerOnIssuer) {
431             token.grantRole(tIssuer, address(issuer));
432             console.log("    granted token.ISSUER_ROLE to TokenIssuer");
433         }
434         if (needOpsOnIssuer) {
435             issuer.grantRole(issuerOps, ops.addr);
436             console.log("    granted issuer.OPS_ROLE to ops");
437         }
438         if (needApprovedRecipient) {
439             issuer.addApprovedRecipient(user.addr);
440             console.log("    added user as approved recipient on the TokenIssuer");
441         }
442         if (needScheduler) {
443             token.grantRole(tScheduler, ops.addr);
444             console.log("    granted token.MULTIPLIER_SCHEDULER_ROLE to ops");
445         }
446         if (needCOps) {
447             c.grantRole(cOps, ops.addr);
448             console.log("    granted compliance.OPS_ROLE to ops");
449         }
450         if (needPOps) {
451             p.grantRole(pOps, ops.addr);
452             console.log("    granted pauseRegistry.OPS_ROLE to ops");
453         }
454         vm.stopBroadcast();
455     }
```

Listing 2.18: script/Smoke.s.sol

Impact If the script contract is public, anyone can derive the private key of the ops wallet and take over operational roles across the token.

Suggestion Revoke the granted roles once the smoke script finishes.

Clarification from BlockSec Note this issue is not within the audit scope. We are disclosing it in accordance with our thorough audit methodology and responsible disclosure principles.

2.1.7 Potential overwriting of a pending multiplier

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The annotation in contract `ERC8056BaseUpgradeable` states that function `_beforeMultiplierUpdate()` can be overridden to prevent overwrites or acceleration. However, contract `ScaledToken` inherits from `ERC8056BaseUpgradeable` without overriding this function to add such a guard. Consequently, function `_setUIMultiplier()` can overwrite a pending schedule without constraints.

```
77 *   - Consider overriding {_beforeMultiplierUpdate} to prevent overwrites or acceleration
78 *   - When overwrites occur, {IERC8056Scheduled-UIMultiplierChangeOverwritten} is emitted
```

Listing 2.19: src/ERC8056/ERC8056BaseUpgradeable.sol

```
419   function _setUIMultiplier(uint256 newMultiplier, uint256 effectiveAtTimestamp) internal virtual
          {
420       require(effectiveAtTimestamp > block.timestamp, "ERC8056: effective time must be in future"
              );
421       require(effectiveAtTimestamp < type(uint256).max, "ERC8056: effectiveAt overflow");
422
423       _validateMultiplier(newMultiplier);
424       _beforeMultiplierUpdate(newMultiplier, effectiveAtTimestamp);
425
426       uint256 currentMult = uiMultiplier();
427
428       // Check if we're overwriting a pending change that hasn't taken effect yet
429       if (block.timestamp < _nextUiMultiplierEffectiveAt && _nextUiMultiplierEffectiveAt != type(
              uint256).max) {
430           _onMultiplierOverwrite(_nextUiMultiplier, _nextUiMultiplierEffectiveAt, newMultiplier,
                  effectiveAtTimestamp);
431       } else if (block.timestamp >= _nextUiMultiplierEffectiveAt) {
432           // Seal any pending change that has already become active
433           _uiMultiplier = _nextUiMultiplier;
434       }
435
436       _nextUiMultiplier = newMultiplier;
437       _nextUiMultiplierEffectiveAt = effectiveAtTimestamp;
438
439       emit UIMultiplierUpdated(currentMult, newMultiplier, effectiveAtTimestamp);
440   }
```

Listing 2.20: src/ERC8056/ERC8056BaseUpgradeable.sol

Impact Overwriting an existing multiplier update may lead to unexpected delays or accelerations.

Suggestion Override the function `_beforeMultiplierUpdate()` with appropriate protection logic.

Feedback from the project The project confirmed that allowing overwriting is intentional because it enables operators to correct an erroneous multiplier or effective time before execution. The project has introduced a `minMultiplierNotice` to enforce a minimum notice period before every multiplier update takes effect.

2.2 Recommendation

2.2.1 Emit issuer transfer events

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `TokenIssuer` emits events for successful `mint()` and `burn()` operations. However, function `transfer()` only emits the underlying ERC-20 `Transfer` event from the token contract. It is recommended that `TokenIssuer` emit a dedicated event so operators and indexers can reconcile issuer inventory movements consistently.

```
43  ///       emitter itself, included so aggregators indexing many issuers can filter without
44  ///       inspecting the log emitter.
45  event Minted(address indexed issuer, address indexed token, address indexed recipient, uint256
        amount);
46
47  /// @notice Emitted on every successful burn routed through this issuer. `from` is either
48  ///       this issuer (burning held inventory) or an approved recipient that pre-approved
49  ///       the issuer to pull tokens.
50  event Burned(address indexed issuer, address indexed token, address indexed from, uint256
        amount);
```

Listing 2.21: src/tokenIssuer/TokenIssuer.sol

```
169 function transfer(address token, address recipient, uint256 amount)
170     external
171     whenNotPaused
172     onlyRole(OPS_ROLE)
173     nonReentrant
174 {
175     if (!isApprovedRecipient(recipient)) revert RecipientNotApproved();
176     IERC20(token).safeTransfer(recipient, amount);
177 }
```

Listing 2.22: src/tokenIssuer/TokenIssuer.sol

Suggestion Emit a `Transferred(address indexed issuer, address indexed token, address indexed recipient, uint256 amount)` event after each transfer executed by `TokenIssuer`.

2.2.2 Remove the redundant codes

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `TokenIssuer`, the event `Executed` is declared but never emitted by any function. Additionally, the function `receive()` is unused, as no flow relies on the contract holding a native balance. It is recommended to remove these redundant codes.

```
40  event Executed(address indexed to, uint256 value, bytes data, bool success);
```

Listing 2.23: src/tokenIssuer/TokenIssuer.sol

```
66    receive() external payable {}
```

Listing 2.24: src/tokenIssuer/TokenIssuer.sol

Suggestion Remove the unused event `Executed` and the function `receive()` from the contract `TokenIssuer`.

2.2.3 Align annotations with implementation

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the project, some annotations do not accurately reflect the implemented behavior. It is recommended to revise them for better code accuracy and readability. Specifically, the following annotations should be aligned with the implementation.

1. In contract `ERC8056BaseUpgradeable`, the annotation on the function `balanceOfUI()` states that it reverts when `balance × multiplier` overflows `uint256`, whereas the product is computed via `Math.mulDiv()` in 512-bit arithmetic and the revert only occurs when the final result exceeds `uint256`.

2. In interface `IScaledUIAmount`, the annotation on the variable `oldMultiplier` of the event `UIMultiplierUpdated` describes it as the previous multiplier, whereas the emitted value is the current multiplier `currentMult`.

3. In contract `TokenIssuer`, function `initialize()` states that the reentrancy guard uses transient storage. However, the contract inherits from `ReentrancyGuard`, which uses storage instead of transient storage.

It is recommended to correct these annotations to reflect the actual implementation or revise the corresponding code.

```
203     * @dev See {IScaledUIAmountBalances-balanceOfUI}.
204     *
205     * Returns the UI-adjusted balance of `account`.
206     *
207     * NOTE: This function deliberately reverts when the multiplier is so extreme
208     * that `balance × multiplier` would overflow `uint256`. Returning 0 in that
209     * case would mislead frontends into showing an empty balance. A revert is a
210     * clearer signal that the UI value is currently unrepresentable.
211     */
212    function balanceOfUI(address account) public view virtual override returns (uint256) {
213        return toUIAmount(balanceOf(account));
214    }
```

Listing 2.25: src/ERC8056/ERC8056BaseUpgradeable.sol

```
13     * @param oldMultiplier The previous multiplier value. A value of 0 indicates
```

Listing 2.26: src/ERC8056/IScaledUIAmount.sol

```
68    function initialize(address admin) external initializer {
69        if (admin == address(0)) revert ZeroAddress();
```

```
70        __AccessControlEnumerable_init();
71        __Pausable_init();
72        // ReentrancyGuard (v5) uses transient storage and needs no initializer.
73
74        _grantRole(DEFAULT_ADMIN_ROLE, admin);
75        _setRoleAdmin(OPS_ROLE, DEFAULT_ADMIN_ROLE);
76    }
```

Listing 2.27: src/tokenIssuer/TokenIssuer.sol

```
102   function _nonReentrantAfter() private {
103       // By storing the original value once again, a refund is triggered (see
104       // https://eips.ethereum.org/EIPS/eip-2200)
105       _reentrancyGuardStorageSlot().getUint256Slot().value = NOT_ENTERED;
106   }
```

Listing 2.28: external_dependency/ReentrancyGuard.sol

Suggestion Align the annotations with the implemented behavior.

2.2.4 Add validations on the symbol

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The token symbol requires additional validation as follows:

1. In the contract `ScaledTokenFactory`, function `renameSymbol()` updates the mapping `tokenBySymbol`, without checking the `token.symbol()`. If misconfigured, the mapping `tokenBySymbol` may become inconsistent with the on-chain symbol of the token. It is recommended to verify that `newSymbol` matches the token's `symbol()` in the function `renameSymbol()`.

2. In contract `ScaledToken`, function `updateSymbol()` accepts any non-empty symbol, while the contract `ScaledTokenFactory` checks the symbol charset and length (i.e., 1–32 characters of `[A-Za-z0-9-]`). This may lead to a symbol that the factory would have rejected being accepted. It is recommended to enforce the same charset check in the function `updateSymbol()`.

```
184   /// @notice Updates the factory's symbol mapping after a token's symbol changes. Caller must
185   ///         have already called {ScaledToken.updateSymbol} on the token itself.
186   function renameSymbol(address token, string calldata oldSymbol, string calldata newSymbol)
187       external
188       onlyRole(DEFAULT_ADMIN_ROLE)
189   {
190       if (bytes(newSymbol).length == 0) revert EmptyString();
191       _assertSymbol(newSymbol);
192       if (tokenBySymbol[oldSymbol] != token) {
193           revert SymbolMismatch(oldSymbol, "token not registered under oldSymbol");
194       }
195       if (tokenBySymbol[newSymbol] != address(0)) revert TokenSymbolAlreadyExists(newSymbol);
196
197       delete tokenBySymbol[oldSymbol];
198       tokenBySymbol[newSymbol] = token;
199
```

```
200         emit SymbolRenamed(token, oldSymbol, newSymbol);
201     }
```

Listing 2.29: src/scaledToken/ScaledTokenFactory.sol

```
139     function updateSymbol(string memory symbol_) external onlyRole(DEFAULT_ADMIN_ROLE) {
140         if (bytes(symbol_).length == 0) revert EmptyString();
141         emit SymbolUpdated(_storedSymbol, symbol_);
142         _storedSymbol = symbol_;
143     }
```

Listing 2.30: src/scaledToken/ScaledToken.sol

Suggestion Verify that `newSymbol` equals the `token.symbol()` in the function `renameSymbol()`, and enforce the `_assertSymbol()` charset check in the function `updateSymbol()`.

2.2.5 Avoid overflow in function `getTokens()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `ScaledTokenFactory`, function `getTokens()` is expected to return up to `limit` tokens starting from `offset`. The implementation computes `offset + limit` before clamping the result to the token array length, so an extremely large `limit` can make the view function revert even when only a small remaining slice should be returned.

```
121     function getTokens(uint256 offset, uint256 limit) external view returns (address[] memory slice
            ) {
122         uint256 total = tokens.length;
123         if (offset >= total || limit == 0) return new address[](0);
124         uint256 end = offset + limit;
125         if (end > total) end = total;
126         uint256 size = end - offset;
127         slice = new address[](size);
128         for (uint256 i = 0; i < size; ++i) {
129             slice[i] = tokens[offset + i];
130         }
131     }
```

Listing 2.31: src/scaledToken/ScaledTokenFactory.sol

Suggestion Compute the remaining length first with `total - offset`, then set the returned size to the smaller value between `limit` and the remaining length.

2.2.6 Add a length check for `newList`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `ComplianceRegistry`, function `replaceSanctions()` switches to a new epoch with a specified sanctions list (i.e., `newList`). However, the function does not validate

the length of `newList`. Consequently, if an empty array is passed, the function does not revert and resets the sanctions list. Such a resetting should instead be handled via the function `clearSanctions()`.

```
126   function replaceSanctions(address[] calldata newList) external onlyOps {
127       uint256 epoch = ++currentEpoch;
128       for (uint256 i = 0; i < newList.length; ++i) {
129           address addr = newList[i];
130           if (addr == address(0)) revert ZeroAddress();
131           if (epochOfUser[addr] == epoch) continue;
132           epochOfUser[addr] = epoch;
133       }
134       emit SanctionsReplaced(newList);
135   }
```

Listing 2.32: src/complianceRegistry/ComplianceRegistry.sol

Suggestion Add a check to ensure the input `newList` is non-empty.

2.2.7 Add a uniqueness check on the `assetCode`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `ScaledTokenFactory`, the function `createToken()` enforces uniqueness on the token symbol but not on `assetCode`, an external instrument code (e.g., an ISIN or a CUSIP) expected to identify the underlying instrument uniquely. As a result, two tokens can be created with the same `assetCode`.

Suggestion Enforce a uniqueness check on `assetCode` in the function `createToken()`.

2.3 Note

2.3.1 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `DEFAULT_ADMIN_ROLE`, `OPS_ROLE`, `ISSUER_ROLE`, `MULTIPLIER_SCHEDULER_ROLE`, and the beacon owner) can conduct sensitive operations, which introduces potential centralization risks. For example, the `ISSUER_ROLE` of contract `ScaledToken` can mint or burn tokens, and `MULTIPLIER_SCHEDULER_ROLE` can schedule UI multiplier updates that rescale every holder's displayed balance. `DEFAULT_ADMIN_ROLE` can rename the token, swap the compliance and pause managers. The `OPS_ROLE` of the contracts `ComplianceRegistry` and `PauseRegistry` can block holders, manage the sanctions list, and pause any token or the entire protocol, directly controlling who may hold or transfer the token. In addition, the owner of the `UpgradeableBeacon` can upgrade the shared `ScaledToken` implementation, upgrading every deployed token at once. 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 and upgrade authorities will be secured by multi-sigs with distributed hardware wallet signers and kept separate from operational keys. The project will also enforce least privilege and key rotation, evaluate tiered timelocks while preserving emergency actions, and establish real-time monitoring and incident-response procedures.

2.3.2 Ensure migration of crucial configurations for existing tokens

Introduced by `Version 1`

Description In contract `ScaledTokenFactory`, function `createToken()` passes the stored `defaultCompliance` and `defaultPauseManager` to new token instances. Functions `setDefaultCompliance()` and `setDefaultPauseManager()` can update these default addresses, while the changes only apply to tokens deployed after the update. Existing tokens remain their previously configured `compliance` and `pauseManager` addresses and must be migrated individually by the token's `DEFAULT_ADMIN_ROLE` via functions `updateCompliance()` and `updatePauseManager()`.

```
23    address public defaultCompliance;
24    /// @notice Default pause registry passed to every new token.
25    address public defaultPauseManager;
```

Listing 2.33: src/scaledToken/ScaledTokenFactory.sol

```
88            "initialize(string,string,string,address,address,address,address[],address[])",
89            _name,
90            _symbol,
91            _assetCode,
92            defaultCompliance,
93            defaultPauseManager,
94            _admin,
95            _issuers,
96            _multiplierSchedulers
97        );
```

Listing 2.34: src/scaledToken/ScaledTokenFactory.sol

```
134       if (_compliance == address(0)) revert ZeroAddress();
135       _assertCompliance(_compliance);
136       emit DefaultComplianceUpdated(defaultCompliance, _compliance);
137       defaultCompliance = _compliance;
138   }
139
140   function setDefaultPauseManager(address _pauseManager) external onlyRole(DEFAULT_ADMIN_ROLE) {
141       if (_pauseManager == address(0)) revert ZeroAddress();
142       _assertPauseManager(_pauseManager);
143       emit DefaultPauseManagerUpdated(defaultPauseManager, _pauseManager);
144       defaultPauseManager = _pauseManager;
145   }
```

Listing 2.35: src/scaledToken/ScaledTokenFactory.sol

Feedback from the project The project confirmed that this behavior is intentional by design.

2.3.3 Dust visibility due to precision loss

Introduced by Version 1

Description In the contract ERC8056BaseUpgradeable, the function balanceOfUI() returns the UI-adjusted balance as rawAmount * uiMultiplier() / MULTIPLIER_DECIMALS, where the result is rounded down. When the function uiMultiplier() returns a value smaller than MULTIPLIER_DECIMALS, a small raw balance (i.e., dust) rounds down to zero. This dust becomes invisible in the UI value, even though the raw balance remains fully preserved in storage. Integrators should rely on the raw balance for accounting and treat the UI amount only as a display value.

Feedback from the project The project confirmed that this is an intended behavior.

2.3.4 Immutable EIP-712 domain

Introduced by Version 1

Description In the contract ScaledToken, the EIP-712 domain is initialized with the original name during initialization, while function updateName() updates the stored name without refreshing the EIP-712 domain. Therefore, functions eip712Domain() and permit() still use the original name after a rename. Off-chain clients should query the function eip712Domain() rather than deriving it from the stored name.

```
100       // EIP-712 domain commits to the *initial* name; later updateName() does not
101       // refresh it. Off-chain clients should query eip712Domain() for the live domain.
102       __ERC20Permit_init(name_);
```

Listing 2.36: src/scaledToken/ScaledToken.sol

Feedback from the project The project confirmed that this is an intended behavior.

2.3.5 Token lifecycle management

Introduced by Version 1

Description In the project, tokens can be listed and issued, and the protocol manages a set of tokens over time. In traditional stock markets, a listed stock can be delisted or retired when it reaches the end of its lifecycle. However, in this project, a listed token remains active once it has been listed. There is no dedicated flow to delist or retire it on-chain. If the backing assets are delisted or retired, the project should use an off-chain mechanism to settle user holdings properly.

Feedback from the project The project confirmed that the protocol does not provide a dedicated on-chain delisting or retirement process. Settlement will be handled off-chain, while functions setMintAllowed(), pauseToken(), and burn() support the wind-down process. The project will document the required sequence in its operational SOP.

2.3.6 Compliance model

Introduced by Version 1

Description Contract `ScaledToken` restricts minting and burning to `ISSUER_ROLE` holders, such as `TokenIssuer`, which applies a recipient whitelist to primary issuance. Secondary transfers rely on contract `ComplianceRegistry`, meaning any address that is not blocked or sanctioned may transfer or receive tokens. Investor eligibility and asset backing are managed off-chain, with no current on-chain proof-of-reserve mechanism. Blocking or sanctioning an address freezes its balance, including preventing transfers and burns. The protocol does not provide a mechanism to burn the frozen tokens.

Feedback from the project The project stated that compliance is layered through approved-recipient controls, off-chain investor verification, and supplementary on-chain blocklist checks. The underlying assets and reserves are maintained and evidenced off-chain, with no current on-chain proof-of-reserve mechanism. Preventing blocked or sanctioned balances from being transferred or burned is intentional freeze semantics. The project will implement a separate burn mechanism if required by future regulations.

2.3.7 Allowances are denominated in raw units

Introduced by `Version 1`

Description In the contract `ScaledToken`, functions `approve()` and `permit()` authorize allowances in raw ERC-20 units without binding the current `uiMultiplier` or UI amount. Because the multiplier may change, the asset-equivalent amount displayed for the same raw allowance may also change over time. Integrators should understand these semantics, use these functions carefully, and calculate the current displayed amount via the function `toUIAmount()`.

Feedback from the project The project confirmed that this is intended behavior. Raw units are the canonical accounting layer for balances, transfers, and allowances. The `uiMultiplier` is used only for display and does not change the raw amount authorized or spendable on-chain.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS