

# Comprehensive Security Audit Report

Project Name: A Fiat-Backed Stablecoin

Audit Date: October 1, 2025

Auditor: Senior Solidity Security Auditor

Scope: Tokens (ERC-20), DAO, Multisigs, Vestings(Solidity ^0.8.26)

Codebase: Provided by file

Blockchain: Ethereum Network

Methodology: Line-by-line manual review, automated analysis, architectural assessment

## EXECUTIVE SUMMARY

### Overall Assessment

The Alliance Foundation smart contract ecosystem represents a sophisticated DeFi protocol implementing a stablecoin (RuBa), governance token (RDAO), vesting mechanisms, and multisig management systems. The codebase demonstrates solid architectural design but contains several critical vulnerabilities that require immediate attention.

### Risk Summary

- Total Contracts Analyzed: 31
- High Vulnerabilities: 1
- Medium Vulnerabilities: 6
- Low/Informational: 7

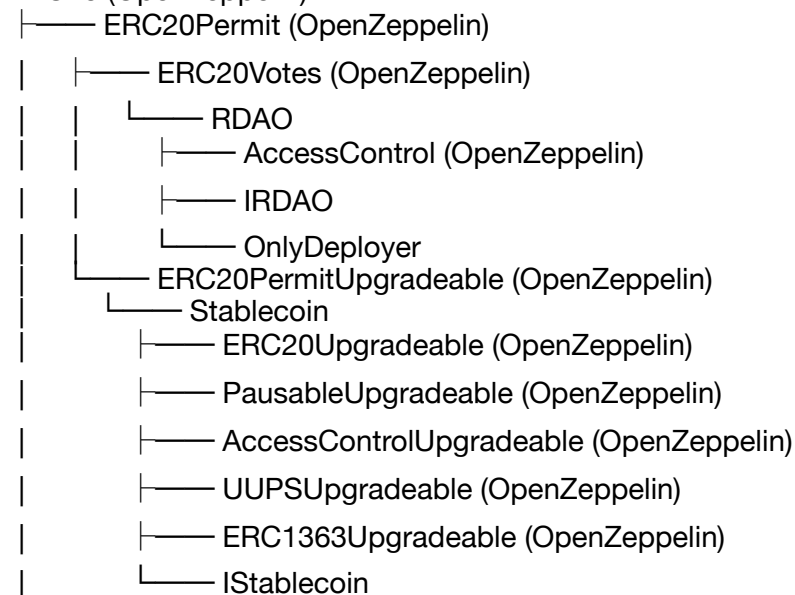
### Deployment Readiness

NOT READY FOR MAINNET - Vulnerabilities must be resolved first.

## CONTRACT INHERITANCE TREE

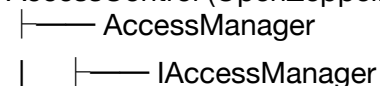
Core Token Contracts

ERC20 (OpenZeppelin)



Access Management Hierarchy

AccessControl (OpenZeppelin)



- | |—— IERC1363Spender (OpenZeppelin)
- | |—— DelegateAllowance
  - | |—— IDelegateAllowance
  - | |—— OnlyDeployer
  - | |—— Initializable (OpenZeppelin)
- |—— Vesting
  - |—— IVesting
  - |—— OnlyDeployer

## Multisig Infrastructure Tree

### MultisigLogic (Base)

- |—— EIP712Upgradeable (OpenZeppelin)
- |—— IMultisigLogic
- |—— ERC165 (OpenZeppelin)
  - |—— OwnerManager
    - |—— IOwnerManager
  - |—— MultisigIssuer
    - |—— AccessControl (OpenZeppelin)
    - |—— IMultisigIssuer
    - |—— Initializable (OpenZeppelin)

### AccessControl (OpenZeppelin)

- |—— MultisigAccessControl
- | |—— OwnerManager
- | |—— IMultisigAccessControl
  - | |—— OnlyDeployer
- |—— MultisigMintManager
  - |—— OwnerManager
  - |—— IMultisigMintManager
    - |—— OnlyDeployer
- |—— MultisigAdmin
  - |—— OwnerManager
  - |—— IMultisigAdmin
    - |—— OnlyDeployer

## Factory and Storage Contracts

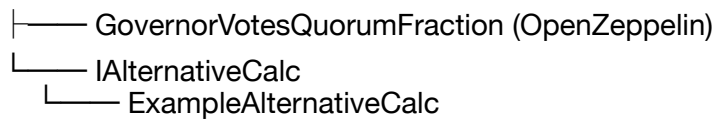
### AccessControl (OpenZeppelin)

- |—— Factory
  - |—— IFactory
    - |—— OnlyDeployer
- |—— StorageIssuer
  - |—— IStorageIssuer
    - |—— OnlyDeployer

## Governance Infrastructure

### Governor (OpenZeppelin)

- |—— Governance
  - |—— GovernorSettings (OpenZeppelin)
  - |—— GovernorCountingSimple (OpenZeppelin)
  - |—— GovernorVotes (OpenZeppelin)



Utility Contracts

Initializable (OpenZeppelin)

└── OnlyDeployer (Base utility for deployment control)

## Inheritance Security Analysis:

Secure Patterns:

- Proper use of OpenZeppelin battle-tested contracts
- Interface-based design reduces coupling
- Clear separation of concerns in inheritance

Potential Issues:

- Complex inheritance chains could lead to unexpected behavior
- Multiple inheritance from OpenZeppelin contracts requires careful state management
- OnlyDeployer inheritance pattern needs validation in proxy contexts

### Areas of Concern:

- Diamond problem potential in complex inheritance
- State variable shadowing risks
- Function selector collisions in complex hierarchies

## OWNERSHIP PRIVILEGES

### Core Token Contracts

Stablecoin.sol

- The ADMIN\_ROLE can pause and unpause the contract.
- The ADMIN\_ROLE can update the whitelist and blacklist status of users.
- The UPGRADER\_ROLE can authorize contract upgrades.
- The MINTER\_ROLE can mint tokens to any address.
- The BURNER\_ROLE can burn tokens from any address with allowance.

RDAO.sol

- The DEFAULT\_ADMIN\_ROLE can grant and revoke minter roles.
- The MINTER\_ROLE can mint governance tokens.
- The deployer can initialize the contract and set the initial supply.

### Access Management Contracts

AccessManager.sol

- The DEFAULT\_ADMIN\_ROLE can grant and revoke all other roles.
- The FACTORY\_ROLE can add and remove issuers.
- The ISSUER\_ROLE can mint and burn RuBa tokens.
- The ISSUER\_ROLE can delegate allowances between issuers.
- The deployer can initialize the contract with critical addresses.

DelegateAllowance.sol

- Inherits access control from parent contracts.
- No direct ownership privileges (abstract contract).

Vesting Contract

Vesting.sol

- The DEFAULT\_ADMIN\_ROLE can grant configurator and access manager roles.
- The CONFIGURATOR\_ROLE can update vesting duration, k parameter, and percent awards.

- The ACCESS\_MANGER\_ROLE can set referrers and manage vesting operations.
- The ACCESS\_MANGER\_ROLE can execute revesting mint and burn operations.
- Referrers can claim rewards for their referred issuers.
- The deployer can initialize the contract with critical parameters.

## **Multisig Infrastructure**

### **MultisigIssuer.sol**

- The MINT\_MANAGER\_ROLE can update mint allowances.
- The AML\_ROLE can update mint pool status.
- Multisig owners can execute minting, burning, and transfer operations.
- Multisig owners can grant and revoke claimer and AML roles.
- Multisig owners can delegate allowances to other issuers.

### **MultisigAccessControl.sol**

- The DEFAULT\_ADMIN\_ROLE can manage validator roles.
- The VALIDATOR\_ROLE must sign all multisig operations.
- Multisig owners can grant and revoke roles on target contracts.
- The deployer can initialize the contract with owners and threshold.

### **MultisigMintManager.sol**

- The DEFAULT\_ADMIN\_ROLE can manage validator roles.
- The VALIDATOR\_ROLE must sign all multisig operations.
- Multisig owners can update mint allowances for issuers.
- The deployer can initialize the contract with owners and threshold.

### **MultisigAdmin.sol**

- The DEFAULT\_ADMIN\_ROLE can manage validator roles.
- The VALIDATOR\_ROLE must sign all multisig operations.
- Multisig owners can update pause status of stablecoins.
- Multisig owners can update whitelist and blocklist status.
- The deployer can initialize the contract with owners and threshold.

### **OwnerManager.sol**

- Multisig owners can add and remove other owners.
- Multisig owners can update the signature threshold.
- Multisig owners can reject nonces to prevent replay attacks.

### **MultisigLogic.sol**

- Provides base signature verification (no direct privileges).
- Validator role must sign all operations.

## **Factory and Storage**

### **Factory.sol**

- The DEFAULT\_ADMIN\_ROLE can update access manager and storage issuer addresses.
- The CONFIGURATOR\_ROLE can update the multisig mint manager address.
- The CREATOR\_ROLE can create new MultisigIssuer contracts.
- The deployer can initialize the factory with implementation addresses.

### **StorageIssuer.sol**

- The DEFAULT\_ADMIN\_ROLE can manage AML and factory roles.
- The AML\_ROLE can add, remove, and update verified issuers.
- The FACTORY\_ROLE can confirm issuer creation.
- The deployer can initialize the contract with role assignments.

## Governance Contracts

Governance.sol

- The governance contract (via proposals) can set alternative calculation accounts.
- Token holders can create proposals and vote on governance decisions.
- No direct admin privileges (decentralized governance).

## Utility Contracts

OnlyDeployer.sol

- The deployer address has exclusive access to onlyDeployer modifier.
- No mechanism to transfer or change deployer address.
- Provides deployment-time access control for inheriting contracts.

## PRE-DEFINED CLAIMS ANALYSIS

✅ Confirmed Capabilities:

- Can Burn: YES - Multiple burn mechanisms in Stablecoin and AccessManager
- Can Mint: YES - Controlled minting in Stablecoin, RDAO, and AccessManager
- Can Blacklist: YES - Blocklist functionality in Stablecoin contract
- Is Upgradeable: YES - UUPS upgradeable pattern in Stablecoin

❌ Not Present:

- Can Lock: NO - No explicit token locking mechanisms
- Can Set Fees (>25%): NO - No fee mechanisms found

## DETAILED VULNERABILITY ANALYSIS

### Stablecoin.sol

**1- Title:** Missing Zero Address Check in Initialize Function

**Severity:** LOW

**Code Location:** ruba/Stablecoin.sol, Line 56

**Description:** The initialize() function accepts an accessControlMultisig parameter without validating that it's not the zero address. If a zero address is passed during deployment, the contract becomes permanently unmanageable since DEFAULT\_ADMIN\_ROLE would be granted to an inaccessible address, preventing all future administrative operations.

**Mitigation:** Add require(accessControlMultisig != address(0), "Zero address not allowed"); before line 50 in the initialize function.

**Status:** OPEN

**2- Title:** Unlimited Pause/Unpause Duration Without Time Constraints

**Severity:** LOW (Mitigated by Upgradeability)

**Code Location:** ruba/Stablecoin.sol, Lines 63-65 (pause function)

**Description:** The pause function allows indefinite contract pausing without maximum duration limits. However, this is mitigated by the contract's upgradeability pattern - malicious or excessive pausing can be corrected through contract upgrades authorized by UPGRADER\_ROLE, providing backstop protection for users.

**Mitigation:** Ensure UPGRADER\_ROLE and ADMIN\_ROLE separation; maintain rapid upgrade capability; implement governance monitoring for pause duration.

**Status:** OPEN

### 3- Title: Missing Timelock Delays for Blacklist Operations

**Severity:** LOW

**Code Location:** ruba/Stablecoin.sol, lines 128-136 (updateBlocklist function)

**Description:** The updateBlocklist() function allows ADMIN\_ROLE to instantly blacklist addresses without any timelock delays or cooling-off periods. This enables immediate freezing of user funds without advance notice, preventing users from taking protective actions and creating potential for abuse or emergency overreach.

**Mitigation:** Implement a time lock mechanism with a 24-48 hour delay for non-emergency blacklisting operations. Add an emergency override function with multi-signature requirements for immediate blacklisting of compromised or legally-ordered addresses only.

**Status:** OPEN

## AccessManager.sol

### 1- Title: Division by Zero in V Parameter Recalculation

**Severity:** HIGH

**Code Location:** AccessManager.sol, Line 306, \_recalcV() function

**Description:** The \_recalcV() function performs division by bth without validation. When multiple issuers burn tokens sequentially, both can reach zero through \_decreaseBth() calls in burnRuba(). This causes a division by zero panic, permanently breaking core reward calculation functionality.

**Mitigation:** Add validation: require(bth > 0, "Cannot recalculate with zero BTH"); before division operation in \_recalcV() function.

**Status:** OPEN

### 2- Title: Unnecessary Dual Allowance Consumption in Managed Transfer

**Severity:** LOW

**Code Location:** AccessManager.sol, Lines 197-198, managedTransferRuba() function

**Description:** The function unnecessarily consumes allowance from both the DelegateAllowance system (line 197) and the ERC20 system (line 198) for the same transfer operation. This

creates operational complexity, requires users to approve both the issuer and AccessManager contract, and causes transaction failures when only one approval exists.

**Mitigation:** Remove line 197 `_spendAllowance(client, issuer, amountRuba)` call as ERC20 allowance check in line 198 provides sufficient security.

**Status:** OPEN

## MultisigIssuer.sol

**1- Title:** Missing Input Validation in Administrative Function

**Severity:** MEDIUM

**Code Location:** MultisigIssuer.sol, Line 58, `updateMintAllowance()` function

**Description:** The `updateMintAllowance()` function accepts any uint256 value without validation, allowing administrators to accidentally or maliciously set mint allowance to zero (disabling minting) or extremely high values (breaking economic assumptions). This poses operational risks and potential system dysfunction despite proper access controls.

**Mitigation:** Add input validation requiring `amount > 0` and `amount <= REASONABLE_MAX_LIMIT`. Consider implementing confirmation mechanisms for extreme values and emitting events showing both old and new values for transparency.

**Status:** OPEN

## StorageIssuer.sol

**1- Title:** Missing Zero Address Validation in Owner Array

**Severity:** MEDIUM

**Code Location:** `storageIssuer/StorageIssuer.sol`, Line 136-139 (`_checkData` function)

**Description:** The `_checkData` function fails to validate that owner addresses in the `IssuerData.owners` array are non-zero. This allows the AML role to store issuer data with `address(0)` owners, causing permanent DoS when Factory attempts to create MultisigIssuer instances, as `_setupOwners` reverts on zero addresses.

**Mitigation:** Add a zero address validation loop in `_checkData` function.

**Status:** OPEN

## MultisigLogic.sol

**1- Title:** Empty Validator Signature Verification Function

**Severity:** MEDIUM

**Code Location:** `multisigs/baseUpgradeable/MultisigLogic.sol`, line 136 (`_checkValidator` function), called from line 105

**Description:** The `_checkValidator()` function is completely empty, providing no validation of the validator signature despite being called in the signature verification process. This allows anyone to

submit arbitrary data as the validator signature, completely bypassing the intended validator security mechanism in multisig operations.

**Mitigation:** Implement proper validator signature verification using ECDSA recovery and role-based validation. Add validator address storage, signature recovery logic, and revert on invalid validator signatures to enforce security requirements.

**Status:** OPEN

**2-Title:** Signature Threshold Logic Inconsistency

**Severity:** MEDIUM

**Code Location:** multisigs/baseUpgradeable/MultisigLogic.sol, line 126  
(\_checkNSignatures function)

**Description:** The \_checkNSignatures function documentation states it should require signatures "greater than or equal to threshold" but implementation uses `len <= _threshold`, requiring `len > threshold`. This forces users to provide one extra signature beyond the intended threshold, causing confusion and usability issues.

**Mitigation:** Change line 126 from `if (len <= _threshold)` to `if (len < _threshold)` to match documented behavior requiring signatures greater than or equal to the threshold value.

**Status:** OPEN

## Vesting.sol

**1- Title:** Division by Zero in Vesting Duration Initialization

**Severity:** MEDIUM

**Code Location:** contracts/vesting/Vesting.sol:70 (initialize function) and contracts/vesting/Vesting.sol:505 (*calcRPS* function)

**Description:** The initialize function accepts `vDuration = 0` without validation. This causes division by zero in `_calcRPS()` when calculating rate-per-second for vesting. All claiming functions (`claimForIssuer`, `claimForReferrer`) and view functions (`getAvailableAmountForIssuer`) become unusable, effectively bricking the contract's core functionality.

**Mitigation:** Add validation in `_setVestingDuration()`: `require(vDuration > 0, "Invalid vesting duration")`. This prevents initialization with zero duration and ensures all rate calculations remain mathematically valid throughout the contract lifecycle.

**Status:** OPEN

**2- Title:** Missing Upper Bound Validation for Percentage Award

**Severity:** MEDIUM

**Code Location:**

- **Function:** `setPercentAward()` (Line 101-103)
- **Root Cause:** `_setPercentAward()` (Line 427-430)
- **Impact Locations:** `claimForReferrer()` (Line 193), `getAvailableAmountForIssuer()` (Line 252)

**Description:** The `setPercentAward()` function lacks upper bound validation, allowing the admin to set percentage values exceeding 100%. This causes `_calcShare()` to return rewards larger than available amounts, triggering arithmetic underflow in claiming functions when calculating `availableAmount - share`, resulting in transaction reverts.

**Mitigation:** Add validation in `_setPercentAward()`: `require(percent <= 100e6, "Percentage cannot exceed 100%")` to ensure referrer rewards never exceed 100% of available vesting amounts.

**Status:** OPEN

**3- Title:** Missing Referrer Validation in Immediate Claim Calculation

**Severity:** LOW

**Code Location:**

- **Function:** `revestingMint()` (Line 129)
- **Root Cause:** `_claimImmediately()` (Line 394)
- **Impact:** Fund loss to `address(0)` when no referrer exists

**Description:** The `_claimImmediately()` function calls `_addReward()` without checking if the referrer exists (`address(0)`). When issuers have no referrer, referrer rewards are still calculated and allocated to `address(0)`, causing permanent fund loss. Issuers receive reduced immediate amounts even without valid referrers.

**Mitigation:** Add referrer validation in `_claimImmediately()`: `if (referrer != address(0)) { imAmountToMint = sh - _addReward(sh, referrer, issuer); } else { imAmountToMint = sh; }`

**Status:** OPEN

## OnlyDeployer.sol

1- **Title:** Unnecessary Initializable Inheritance in OnlyDeployer

**Severity:** INFORMATIONAL

**Code Location:** `utils/OnlyDeployer.sol:4,6`

**Description:** The `OnlyDeployer` contract inherits from OpenZeppelin's `Initializable` but doesn't require upgradeable functionality. This creates unnecessary dependency and potential confusion since the contract uses constructor-based initialization rather than proxy initialization patterns, making the `Initializable` inheritance redundant.

**Mitigation:** Remove the `Initializable` import and inheritance. Change line 6 from `contract OnlyDeployer is Initializable` to `contract OnlyDeployer`. This eliminates unnecessary dependency without affecting functionality.

**Status:** OPEN

## All Contracts

1- **Title:** Floating Pragma Version

**Severity:** LOW

**Code Location:** All Core Contracts

**Description:** All contracts use a floating pragma ^0.8.26 instead of locking to a specific compiler version. The contracts use pragma solidity ^0.8.26, which allows compilation with any version from 0.8.30 up to (but not including) 0.9.0. This creates deployment inconsistencies where different compiler versions may produce different bytecode, introducing subtle bugs or behavioral changes across deployments.

**Mitigation:** Lock the pragma to a specific compiler version using pragma solidity 0.8.26; to ensure consistent compilation and deployment behavior across all environments.

**Status:** OPEN

## GAS OPTIMIZATION OPPORTUNITIES

1. Use unchecked blocks for safe arithmetic operations
2. Pack struct variables to optimize storage usage
3. Cache storage variables in memory when used multiple times
4. Use calldata instead of memory for function parameters when possible
5. Implement batch operations for multiple similar transactions

## FINAL VERDICT

Overall Security Score: 6.5/10

Deployment Recommendation: **✗** DO NOT DEPLOY TO MAINNET

The Alliance Foundation smart contracts show promise with solid architectural design, but vulnerabilities must be addressed before any production deployment that could lead to fund loss or protocol manipulation.

Next Steps:

1. Address all issues
2. Implement a comprehensive testing suite
3. Consider additional external security audits
4. Establish ongoing security monitoring procedures

## DISCLAIMER

This audit report is not the final deployed version of the contract. So, any changes made after the audit will be considered out of the scope of the audit, and we will not be responsible for any changes related to that.

**Audit Completed: October 1, 2025**

**Report Version: 1.0**

**Auditor Signature:**

