Smart Contract Audit Report

AccessControlUpgradable

January 1st, 2025

(Amended on January 30th, 2025)

Index

Summary

Audit report was prepared by an independent auditor for the AccessControlUpgradable smart contracts. The audit aimed to identify vulnerabilities, optimize gas efficiency, and ensure compliance with Solidity best practices.

Audit Process & Delivery

Two (2) independent experts performed an unbiased and isolated audit of the code below. Client discussions and debrief occurred between 13–17 December 2024, followed by the audit process from 24–31 December 2024, with the final results presented here.

Audited Files in scope

- **InitializableAccessControl**
- **InitializableAccessControlCore.sol**
- **UpgradeableAccessControl.sol**
- **UpgradeableAccessControlCore.sol**

Client Documents/Resources

The client provided a GitHub repository as the primary resource for this audit. The audit was conducted on commit `34bd526b2ba15ea3fb8f91f3be6a509309f2f0c4`

Intended Behavior

The following points outline the intended behavior of the dApp, derived from the README documentation provided in the GitHub repository. These behaviors reflect the principles and implementation details described in the repository, ensuring the dApp achieves its goals of modularity, security, and decentralization through Role-Based Access Control (RBAC).

Role-Based Access Control (RBAC):

- Provides modular and pluggable architecture for dApps.
- Uses a bitmask-based system to manage **features** (global permissions) and **roles** (user-specific permissions).
- Supports up to 256 roles and features with lightweight implementation.

Phased Control Lifecycle:

- Contracts start fully controlled by deployers/admins.
- Transition through phases to achieve decentralization by revoking admin privileges and relying on a DAO/community.

Upgradeable Architecture:

- Follows OpenZeppelin's UUPS Proxy pattern for upgradeability.
- Introduces `ROLE_UPGRADE_MANAGER` to control and authorize upgrades, which can eventually be revoked for full decentralization.

RBAC API:

- Enables checking and enforcing role-based access for users (`isSenderInRole`) and features (`isFeatureEnabled`).
- Supports granting/revoking roles and enabling/disabling features.

Initializable Contracts:

- Uses `_postConstruct` initialization for compatibility with proxy deployments.
- Allows setting initial roles and features during deployment.

Security and Flexibility:

- Supports gradual activation of contract functionalities via feature flags.
- Ensures restricted access to sensitive operations using roles.
- Allows eventual decentralization by revoking administrative roles.

Issues Found

Critical

No Critical issues were found.

Major

No Major issues were found.

Minor

1. ### Insufficient Event Logging in `RoleUpdated`

   Issue: The `RoleUpdated` event in InitializableAccessControlCore.sol does not log the `msg.sender` (executor of the update).

   Impact: Without `msg.sender`, it becomes difficult to trace the origin of role updates during debugging or audits, potentially obscuring accountability.

   Recommendation: Modify the `RoleUpdated` event to include `msg.sender` as a parameter. This addition will improve traceability and make audits more comprehensive.

   Amended (January 30th, 2025): The RoleUpdated event in AccessControlCore was enhanced to include the caller's address (msg.sender). This improvement ensures better traceability of role updates. The issue is no longer present as of commit 28609d3f2707820ed3ca4bae47d7cde113088d03

2. **No Check for role != 0 in updateRole**

   Issue: Allowing role = 0 could unintentionally revoke all permissions from an operator.

   Impact: Allowing role = 0 without explicit validation or logging can unintentionally revoke all permissions from an operator, potentially disrupting operations if done accidentally.This could lead to administrative errors, such as removing critical permissions during a misconfigured update, which may require additional intervention to restore functionality.

   Recommendation: Add a check or explicit logging for such actions.

   Amended (January 30th, 2025): The client clarified that the updateRole function is intentionally designed to handle adding, updating, and revoking roles in a single function to avoid additional public functions. This approach was deemed valid, and the justification was accepted.

Notes

1. **Version Constraints with Known Issues:**

   Issue: The contracts currently use `>=0.8.4`, which might miss recent compiler bug fixes and improvements.
   Impact: Using an older version could expose contracts to previously resolved issues or inefficiencies.
   Recommendation: Consider upgrading all contracts to the latest stable Solidity version (`^0.8.26`) for enhanced security and functionality.

   Amended (January 30th, 2025): The client updated the compiler version to 0.8.28 in hardhat.config.js instead of individual files. This decision was justified by the need to maximize compatibility for RBAC-based applications and serve as a Solidity library. The issue is mitigated and no longer present as of commit 28609d3f2707820ed3ca4bae47d7cde113088d03

2. Visibility Optimization for Gas Savings:

Issue: Several functions are declared as `public`, though they could benefit from more specific visibility (`external` or `internal`). Some are not called within the contract or inherited contracts, meaning they could be marked as `external`. Similarly, functions that are not supposed to be called from outside can be marked internal.

Impact: Keeping functions `public` when they can be `internal` or `external` increases gas usage unnecessarily due to argument memory copying or visibility checks.

Recommendation: Consider declaring functions as external or internal as per the requirement. These are a few functions that are identified for being marked as external: isFeatureEnabled, isSenderInRole, isOperatorInRole, getInitializedVersion, and other such functions.

Amended (January 30th, 2025): Functions were updated to external where applicable. This optimization reduces gas usage for external calls. The issue is no longer present as of commit 28609d3f2707820ed3ca4bae47d7cde113088d03

3. **Potential Improvement for FULL_PRIVILEGES_MASK Assignment**:

   Issue: The FULL_PRIVILEGES_MASK grants super-admin privileges. Its assignment is gated by the ROLE_ACCESS_MANAGER role, ensuring only authorized accounts can make changes.

   Impact:While adequate safeguards exist for trusted environments, the lack of additional mechanisms for assigning such critical roles could pose risks in systems prioritizing strict decentralization.A single-point assignment may lead to misuse or errors in decentralized setups, potentially compromising the system's integrity.

   Recommendation: Consider implementing a multi-signature or governance model for assigning critical roles to further enhance security and decentralization.

   Amended (January 30th, 2025): The client clarified that the system is designed as a temporary RBAC solution transitioning to DAO governance. This approach aligns with their roadmap, and the justification was accepted.

4. **Test Coverage:**

   Issue: While test cases exist, branch coverage is not 100%.

   Impact: Incomplete test coverage leaves potential edge cases untested, increasing the risk of bugs going unnoticed in production.

   Recommendation: Consider Conducting a detailed review of uncovered branches and adding tests to ensure complete coverage, especially for critical functions.

   Amended (January 30th, 2025): Additional test cases were added to cover previously uncovered branches, achieving 100% branch coverage. This ensures all functionalities, including edge cases, are rigorously tested. The issue is no longer present as of the commit 28609d3f2707820ed3ca4bae47d7cde113088d03

Closing Summary

Upon audit of the AccessControlUpgradable smart contract, it was observed that the contracts contain minor issues, along with several areas of notes. The audit process involved a comprehensive manual review supported by automated analysis using the Slither tool. Each identified issue was carefully validated, and the code was found to be of high quality, featuring well-structured logic, robust access control mechanisms, proper commenting, and adherence to best practices. Comprehensive test cases were provided, achieving 100% coverage for functions and lines of code, although branch coverage was not fully achieved, leaving some edge cases untested.

We recommend that minor issues should be resolved. Resolving the areas of notes is up to the client’s discretion, as the notes refer to potential improvements in the operations of the smart contract.

Disclaimer

This audit is not a security warranty, investment advice, or an endorsement of the AccessControlUpgradable contract. This audit does not guarantee the complete security or correctness of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The audit was conducted on commit `34bd526b2ba15ea3fb8f91f3be6a509309f2f0c4`. All findings and recommendations in this report apply specifically to the codebase at this commit.

The individual audit reports are anonymized and combined during a debrief process to provide an unbiased delivery and protect the auditors from legal and financial liability.