Smart Contract Audit Report

AccessControl

December 23rd, 2024

(Amended on January 20th, 2025)

Index

Index..... 2
Summary..... 3
Audit Process & Delivery..... 3
Audited Files in scope..... 3
Client Documents/Resources..... 3
Intended Behavior..... 4
Access Management Framework:..... 4
Feature and Role Definitions:..... 4
Lifecycle Phases:..... 4
Decentralization Capability:..... 5
Role and Permission Management:..... 5
Security Best Practices:..... 5
Error Handling:..... 5
Backward Compatibility:..... 6
Operational Usage:..... 6
Issues Found..... 6
Major..... 6
Minor..... 8
Notes..... 9
Closing Summary..... 12
Disclaimer..... 12

Summary

Audit report was prepared by an independent auditor for the AccessControl 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 18–23 December 2024, with the final results presented here.

Audited Files in scope

- **AccessControl.sol**
- **AccessControlCore.sol**
- **AdapterFactory.sol**
- **OwnableToAccessControlAdapter.sol**

Client Documents/Resources

The client provided a GitHub repository as the primary resource for this audit. The audit was conducted on commit `d30f06fe61307f2d8bcbd594ac0e34fc4e695abd`.

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).

Access Management Framework:

- RBAC serves as a parent contract to enable modular, pluggable architecture by managing roles and permissions.
- It provides APIs to check global and user-specific permissions for executing specific operations.

Feature and Role Definitions:

- **Features**: Control the availability of public functions, allowing enablement/disablement during deployment, setup, and optionally during operations.
- **Roles**: Govern access to restricted functions, intended for a limited set of maintainers.

Lifecycle Phases:

- **Initial Deployment**: Deployer holds permissions to set up and verify the contract.
- **Setup Completion**: Deployer revokes permissions, enabling decentralized control by multisig admins or DAOs, ensuring the contract operates securely in a community-driven model.

Decentralization Capability:

- Contracts can transition from centralized deployment to fully decentralized governance or no governance at all.
- Access manager roles can revoke their own permissions, allowing for ungoverned operation.

Role and Permission Management:

- Permissions are represented as 256-bit integers, with each bit corresponding to a specific role.
- The `ROLE_ACCESS_MANAGER` (bit 255) allows granting/revoking permissions and enabling/disabling features.

Security Best Practices:

- Public mutative functions are gated by feature flags for controlled enablement.
- Restricted access functions ensure only specific roles can execute them.

Error Handling:

- Access checks failing due to insufficient permissions throw an `AccessDenied` error. Custom errors or messages can also be defined.

Backward Compatibility:

- Allows migration from OpenZeppelin's `Ownable` model using an adapter to map `Ownable` functions to RBAC roles.

Operational Usage:

- Features and roles can be queried and managed via APIs, allowing detailed tracking and evaluation of contract permissions.
- `RoleUpdated()` events enable auditing changes to roles and permissions.

Issues Found

Major

1. **Low-Level Call in `execute(bytes)` (OwnableToAccessControlAdapter)**
   Issue: The execute(bytes) function in OwnableToAccessControlAdapter uses a low-level call without additional safety mechanisms. This can result in unexpected behavior if the target contract behaves maliciously or unexpectedly.

   Impact: While the call is validated with a require(success, "Execution failed"), it lacks other safety mechanisms provided by OpenZeppelin's Address.functionCall.
   For instance, Address.functionCall ensures the target is a contract, providing additional safety.

Recommendation: Replace the low-level call with OpenZeppelin's safer alternative:
bytes memory result = Address.functionCall(target, data, "Low-level call failed");

Amended (January 20th, 2025): A new function __requireSuccessfulCall was added in OwnableToAccessControlAdapter to handle low-level call errors and propagate meaningful error messages. The client justified not using OpenZeppelin's Address.functionCall to avoid adding unnecessary dependencies, opting instead to port only the relevant functionality (error propagation and address.code.length validation). The issue is no longer present as of commit 06f48a62d6ab6e418aaa8e179e0d43be99575e40.

2. Validation of `targetAddress` in `deployNewOwnableToAccessControlAdapter` (AdapterFactory)

Issue: The `targetAddress` parameter in deployNewOwnableToAccessControlAdapter is not validated for being a valid contract address or non-zero.

Impact: An invalid or zero targetAddress could result in the deployment of an unusable adapter contract. In decentralized systems, such validations are crucial to avoid misconfigurations.

Recommendation: Add checks to validate the targetAddress:
require(targetAddress != address(0), "Zero address not allowed");
require(targetAddress.code.length > 0, "Address is not a contract");

Amended (January 20th, 2025): Explicit checks for targetAddress were added in AdapterFactory to ensure the address is not zero and points to a deployed contract. This addresses the concern and prevents invalid addresses. The issue is no longer present as of commit 06f48a62d6ab6e418aaa8e179e0d43be99575e40.

Minor

1. Insufficient Event Logging in RoleUpdated

Issue: The RoleUpdated event in AccessControlCore.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 20th, 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 06f48a62d6ab6e418aaa8e179e0d43be99575e40.

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 20th, 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.

3. **State Variables Could Be Declared immutable (OwnableToAccessControlAdapter)**:
   Issue: The target variable is set in the constructor but is not declared as immutable.

   Impact: Not declaring the target variable as immutable results in higher gas costs when accessing the variable.It also leaves room for unintended changes during the contract's lifecycle, which could potentially introduce risks in critical operations.

   Recommendation: Consider declaring it as immutable to reduce gas costs and prevent accidental changes.

   Amended (January 20th, 2025): The target variable in OwnableToAccessControlAdapter was declared as immutable. This ensures it cannot be changed after deployment and improves gas efficiency. The issue is no longer present as of commit 06f48a62d6ab6e418aaa8e179e0d43be99575e40.

Notes

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

   Issue: Solidity versions used (>=0.8.4, ^0.4.11) have known vulnerabilities.

   Impact: Using Solidity versions with known vulnerabilities exposes the contracts to potential exploits or undefined behavior, especially in edge cases.Some of these vulnerabilities might impact the reliability of compiled bytecode or result in runtime errors, depending on specific scenarios.
   It also limits compatibility with tools and updates that rely on the latest Solidity features and fixes.

   Recommendation: Upgrade to a safer Solidity version (>=0.8.20).

Amended (January 20th, 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 06f48a62d6ab6e418aaa8e179e0d43be99575e40

2) **Visibility Optimization for Gas Savings:**

Issue: Several functions are declared as public but are not appropriately scoped. 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 as 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: isOperatorInRole, isSenderInRole, isFeatureEnabled, and other such functions.

Amended (January 20th, 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 06f48a62d6ab6e418aaa8e179e0d43be99575e40.

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 20th, 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.

Closing Summary

Upon audit of the AccessControl smart contract, it was observed that the contracts contain major and 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, and ensuring all functionalities were rigorously tested.

We recommend that major and 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 AccessControl 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 `d30f06fe61307f2d8bcbd594ac0e34fc4e695abd`. 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.