Security Audit Report

AllUnity

v1.0

July 11, 2025

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by AllUnity GmbH to perform a security audit of Audit of the AullUnity stable coin smart contract.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/All-Unity/smart-contracts
Commit	`f5c7927695d7c148eebba88ce9d9f46f28bc3ce7`
Scope	The scope is restricted to the contracts located in `ethereum/ethereum-contracts/contracts`
Fixes verified at commit	`9d56d76054128c5b7fe5b5258fefa829abc21839` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The AllUnity protocol implements an ERC-20 token with role-based access control.

The protocol consists of three main contracts: AllUnity (the token contract with ERC-20, ERC-1363, and permit functionality), BlacklistManagement (maintains addresses forbidden from transacting), and a proxy contract following the ERC-1967 standard.

The architecture uses designated roles (admin, pauser, minter, burner, blacklister) to manage permissions and includes mechanisms to pause transfers, mint and burn tokens, and block specific addresses from sending or receiving tokens.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Low-Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium**	-
Test coverage	**High**	`hardhat coverage` reports a test coverage of `100%`

Summary of Findings

No	Description	Severity	Status
1	Missing role segregation leads to accumulation of privileges	Minor	Resolved
2	Missing storage gap in UUPS proxy pattern undermines upgrade safety	Minor	Resolved
3	Centralization risks	Minor	Acknowledged
4	Lack of batch removal function for blacklisted addresses	Informational	Resolved

Detailed Findings

1. Missing role segregation leads to accumulation of privileges

Severity: Minor

In `ethereum/ethereum-contracts/contracts/AllUnity.sol:46-49`, the AllUnity contract lacks proper role segregation, assigning multiple roles to a single address.

Specifically, one address holds both `DEFAULT_ADMIN_ROLE` and `PAUSER_ROLE`, while another possesses `MINTER_ROLE` and `BURNER_ROLE`.

This design results in lack of role segregation, if either address is compromised, a malicious actor could gain the ability to arbitrarily mint or burn tokens, freeze contract functionality, or modify access control configurations.

Such privilege consolidation violates the principle of least privilege and introduces governance and operational risks.

Recommendation

We recommend implementing strict role segregation policies by distributing critical roles, such as `DEFAULT_ADMIN_ROLE`, `PAUSER_ROLE`, `MINTER_ROLE`, and `BURNER_ROLE`, across distinct addresses.

Status: Resolved

2. Missing storage gap in UUPS proxy pattern undermines upgrade safety

Severity: Minor

The AllUnity contract utilizes the UUPS (Universal Upgradeable Proxy Standard) proxy upgrade pattern but omits a reserved storage gap.

Specifically, the contract defines state variables such as `m_decimals` without allocating storage slots for potential future variables.

This absence of a storage gap can lead to storage collisions during future upgrades, potentially resulting in corrupted state variables or unexpected contract behavior, thereby compromising the integrity and reliability of the upgrade process.

Recommendation

We recommend reserving an appropriately sized storage gap in the contract layout when implementing the UUPS pattern.

Status: Resolved

3. Centralization risks

Severity: Minor

In AllUnity and BlacklistManagement contracts, there are several privileged roles that create significant centralization risks:

- `DEFAULT_ADMIN_ROLE` can upgrade contracts with no timelock, allowing immediate deployment of potentially malicious code
- `BLACKLISTER_ROLE` can add any address to a blacklist without restrictions
- When combined, `BLACKLISTER_ROLE` and `BURNER_ROLE` enable token seizure as `burnFrom` bypasses allowance checks for blacklisted addresses
- `PAUSER_ROLE` can freeze all protocol operations by pausing the contract
- `MINTER_ROLE` and `BURNER_ROLE` can mint and burn tokens without any limits
- The `burnFrom` function in AllUnity specifically skips allowance validation for blacklisted addresses, enabling forced token burning

Recommendation

We recommend enforcing strict key management, the usage of multi-signature accounts and evaluating the removal of the allowance bypass in `burnFrom` for blacklisted addresses.

Status: Acknowledged

4. Lack of batch removal function for blacklisted addresses

Severity: Informational

In `ethereum/ethereum-contracts/contracts/BlacklistManagement.sol:40-46`, the contract provides a `batchAddBlacklist` function that allows adding multiple addresses to the blacklist in a single transaction, but lacks an equivalent function for removing multiple addresses.

Currently, blacklisted addresses can only be removed individually through the standard AccessControl's role revocation methods, which creates an asymmetric design and operational inefficiency when bulk operations are needed.

Recommendation

We recommend implementing a `batchRemoveBlacklist` function that allows authorized roles to remove multiple addresses from the blacklist in a single transaction.

Status: Resolved