QuillAudits

AUDIT REPORT

September 2025

For

AdLunam

Table of Content

Executive Summary 03
Number of Security Issues per Severity 05
Summary of Issues 06
Checked Vulnerabilities 07
Techniques and Methods 08
Types of Severity 11
Types of Issues 12
Severity Matrix 13
Low Severity Issues 14
1. No validation that initialRecipient is not the zero address. 14
Informational Issues 15
2. Floating Pragma Version 15
Functional Tests 16
Automated Tests 16
Threat Model 17
Note to Users/Trust Assumptions 18
Closing Summary & Disclaimer 19

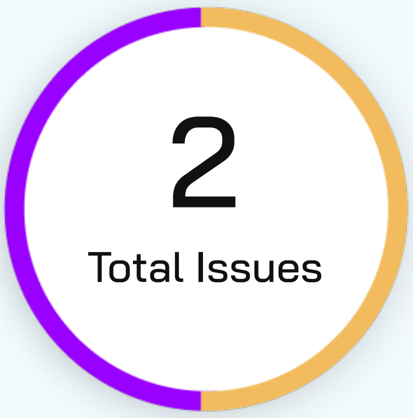
Executive Summary

Project Name	LUNAM
Protocol Type	ERC20 Token
Project URL	https://adlunam.cc
Overview	LUNAM is a ERC20 token with fixed-supply and EIP-2612 permit support for gasless approvals. It mints a total of 1,000,000,000 LUNAM (18 decimals) once at deployment to a specified recipient, typically a multisig wallet. No additional minting or burning functionality is provided, ensuring a hard-capped supply.
Audit Scope	The scope of this Audit was to analyze the Lunam Smart Contracts for quality, security, and correctness.
Source Code Link	https://github.com/Adlunamdev/LunamToken/blob/main/lunamtoken.sol
Branch	Main
Commit Hash	d1f85a4731d454c823cf8b5bc70e4b33b418ac79
Contracts in Scope	Lunam.sol
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	23rd September 2025
Updated Code Received	29th September 2025
Review 2	29th September 2025 - 30th September 2025
Fixed In	4c66f2be93e0f41db226b07f94cc49275efcb50b

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity



🟥	Critical	0 (0.0%)
🟥	High	0 (0.0%)
🟧	Medium	0 (0.0%)
🟨	Low	1 (50.0%)
🟪	Informational	1 (50.0%)

Severity

Issues

	Critical	High	Medium	Low	Informational
Open	0	0	0	0	0
Acknowledged	0	0	0	0	0
Partially Resolved	0	0	0	0	0
Resolved	0	0	0	1	1

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	No validation that initialRecipient is not the zero address.	Low	Resolved
2	Floating Pragma Version	Informational	Resolved

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls

- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw
- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ Unsafe type inference
- ✓ Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts:

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Static Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

🟥 Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

🟥 High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

🟧 Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

🟨 Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

🟪 Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

Types of Issues

Open Security vulnerabilities identified that must be resolved and are currently unresolved.	**Resolved** These are the issues identified in the initial audit and have been successfully fixed.
Acknowledged Vulnerabilities which have been acknowledged but are yet to be resolved.	**Partially Resolved** Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Severity Matrix

Impact

Likelihood

	🟥 High	🟧 Medium	🟨 Low
🟥 High	Critical	High	Medium
🟧 Medium	High	Medium	Low
🟨 Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Low Severity Issues

No validation that initialRecipient is not the zero address.

Resolved

Description

The constructor calls _mint(initialRecipient, INITIAL_SUPPLY) without first validating initialRecipient. OpenZeppelin's_mint will revert if address(0) is passed, but relying on that implicit revert is less clear and reduces readability. An explicit guard documents intent and produces a clearer revert message.

Recommendation

Add an explicit require with a clear error message at the start of the constructor:

```
require(initialRecipient != address(0), "LUNAM: initial recipient is zero");

```

Informational Issues

Floating Pragma Version

Resolved

Description

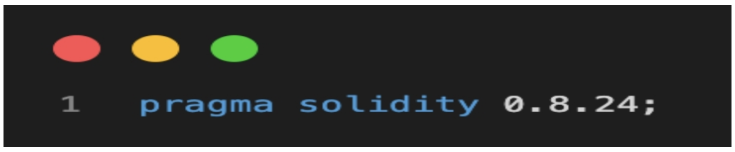
Currently Contract uses pragma solidity ^0.8.24; This allows compilation with 0.8.24 and higher minor versions (0.8.25, 0.8.26, ...), which could introduce undetected changes or vulnerabilities in newer compiler releases.

Floating pragmas can lead to:

- Different compilation results across environments
- Potential introduction of bugs in newer compiler versions
- Inconsistent bytecode between deployments
- Difficulty in reproducing exact deployment conditions

Recommendation

Use a fixed pragma version for production deployments:



```
pragma solidity 0.8.24;
```

Functional Tests

Some of the tests performed are mentioned below:

- ✓ should return name
- ✓ should return symbol
- ✓ should return decimals = 18
- ✓ should have correct total supply
- ✓ should assign the entire initial supply to recipient
- ✓ should transfer tokens between accounts
- ✓ should fail when transferring more than balance
- ✓ should emit Transfer event on transfer
- ✓ should allow transferring 0 tokens
- ✓ should approve and allow transferFrom
- ✓ should fail transferFrom when allowance is insufficient
- ✓ should update allowance after transferFrom
- ✓ should emit Approval event on approve
- ✓ should increase allowance correctly
- ✓ should decrease allowance correctly
- ✓ should fail decreaseAllowance below 0
- ✓ should revert when transferring to address(0)
- ✓ should revert when minting to address(0) in constructor (zero recipient)
- ✓ should maintain invariant: sum of balances == totalSupply
- ✓ should preserve fixed total supply (no further minting possible)

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Threat Model

Contract	Function	Threats
LUNAMToken	Constructor	1. initialRecipient set to address(0) (implicit revert but unclear) → deployment confusion. 2. Single-key initialRecipient custody risk (full-supply hot-wallet compromise). 3. Wrong recipient address supplied by deployer (human error).

Note to Users/Trust Assumptions

Immutable Supply Design:

- The total supply of 1,000,000,000 LUNAM tokens is permanently fixed
- No additional tokens can ever be minted or created
- No tokens can be burned by the contract (unless sent to burn addresses)

Centralization Risks:

- The initialRecipient (deployer/multisig) receives 100% of the token supply at deployment
- Token distribution depends entirely on the actions of this initial recipient
- No built-in distribution mechanisms or vesting schedules exist in the contract

Trust Dependencies:

- Complete trust is required in the initial recipient for fair distribution
- The initial recipient could theoretically dump the entire supply at once
- No governance mechanisms or community controls are implemented

Technical Trust Points:

- Contract relies on OpenZeppelin's battle-tested ERC20 and ERC20Permit implementations
- No admin functions or upgrade mechanisms exist (fully immutable once deployed)
- Permit functionality allows gasless approvals (EIP-2612 standard)

Due Diligence Recommendations for Users:

- Verify the deployer's identity and reputation before investing
- Check token distribution plans and transparency commitments
- Monitor initial recipient's wallet for distribution patterns
- Confirm the contract address matches official announcements

Closing Summary

In this report, we have considered the security of LUNAM Token. We performed our audit according to the procedure described above.

The contract has a sound design for a fixed-supply token with permit functionality, but contains a Very minor issue. Once fixed, it follows good practices for immutable token contracts.

Disclaimer

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With seven years of expertise, we've secured over 1400 projects globally, averting over $3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

QuillAudits

7+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

September 2025

For

AdLunam

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com