CASHIVA

# Cashiva Audit Report For LanTal Mining

Revision Date: 2025-05-12

## Table of Contents

# 1. Executive Summary

## 1.1. Project Introduction

This report details the findings of a smart contract security audit performed by Cashiva Community LLC on the LMAU contracts, provided by LanTal Mining LLC. The LMAU token is designed to represent legal title to physically-backed gold, implemented via the audited smart contracts as an ERC20-compliant token with additional features including fees, freezability, and upgradeability.

## 1.2. Audit Objectives & Scope

The primary objective of this audit was to identify potential security vulnerabilities, design flaws, and deviations from best practices within the provided Solidity smart contract code, designed to manage the token on-chain. The scope was limited to the smart contracts associated with the commit hash 84e56aa2b3cf87e95d823ef59f7f1d28b1e9d08a. The audit does not cover verification of the off-chain physical asset backing or associated legal structures.

## 1.3. Overall Security Assessment

The LMAU contracts leverage OpenZeppelin Upgradeable contracts, which is a strong foundational choice for the on-chain implementation.

A significant medium severity finding (LMAU-M-001) was identified concerning how the `Feeable` contract interacts with ERC20 storage. While not an immediate external exploit, this flaw poses a considerable risk to the contract's long-term stability and data integrity, particularly during routine dependency upgrades, and could lead to severe consequences if triggered. It requires prioritized attention. Other medium severity findings relate to the scope of pausable functionality and the absence of an on-chain maximum fee rate cap.

A core characteristic of the contract is the extreme concentration of power in the `owner` role, which has unilateral control over all critical aspects of the token system. This presents significant centralization risks, as the contract's integrity and user assets are wholly dependent on the security and behavior of the entity controlling the `owner` address.

Addressing all identified findings, with particular emphasis on LMAU-M-001 due to its potential impact, and carefully considering mitigations for the extensive owner powers, is essential to ensure the long-term security, reliability, and upgradeability of the on-chain system.

## 1.4. Summary of Findings

| ID | Title | Severity | Status |
|---|---|---|---|
| LMAU-M-001 | Direct ERC20 Storage Manipulation in `Feeable` Contract with Potential for Severe Impact | Medium | Acknowledged |
| LMAU-M-002 | Pausability Scope on Mint and Redeem Functions | Medium | Acknowledged |
| LMAU-M-003 | Absence of an On-Chain Maximum Fee Rate Cap | Medium | Acknowledged |
| LMAU-I-001 | Visibility of Fee Calculation Logic | Informational | Acknowledged |

## 2. Disclaimer

This audit report is provided for informational purposes only and is based on the code provided to Cashiva Community LLC at a specific point in time (commit hash `*84e56aa2b3cf87e95d823ef59f7f1d28b1e9d08a*`). Smart contract security is a complex and evolving field; an audit does not guarantee the absence of all vulnerabilities.

**Crucially, the scope of this report is limited to the smart contract code; it does not constitute a financial audit or verification of any off-chain assets, processes, or legal claims referenced in project documentation.**

Cashiva Community LLC makes no warranties, express or implied, regarding the complete security of the audited code or its fitness for any particular purpose. The client is solely responsible for the deployment, maintenance, and operation of the smart contracts, and for the veracity and execution of any off-chain procedures. This report should not be considered investment advice.

## 3. About Cashiva Community LLC

Cashiva Community LLC is a company that specializes in blockchain technologies, DeFi development, and smart contract auditing. We are dedicated to fostering a more secure and reliable Web3 ecosystem by providing comprehensive security assessments and development expertise.

Our team is composed of seasoned blockchain professionals, security researchers, and smart contract engineers with deep expertise across a range of domains, including:

- **Smart Contract Auditing:** In-depth analysis of Solidity and other smart contract languages to identify vulnerabilities, logic flaws, and gas optimization opportunities.
- **DeFi Protocol Development & Security:** Extensive experience in designing, building, and securing complex decentralized finance applications, including lending platforms, DEXs, yield farming protocols, and more.
- **Blockchain Technology & Architecture:** Profound understanding of core blockchain principles, consensus mechanisms, cryptographic primitives, and various Layer 1 and Layer 2 solutions.
- **Exploitation Techniques & Mitigation Strategies:** Up-to-date knowledge of common and novel attack vectors targeting smart contracts and blockchain systems, and best practices for their prevention.
- **Security Best Practices & Standards:** Adherence to industry-leading security guidelines and development standards.

We employ a meticulous audit methodology that combines manual line-by-line code review, automated static and dynamic analysis tools, and conceptual logic evaluation to deliver thorough and actionable security reports. Our goal is to empower our clients with the insights and recommendations needed to build and deploy secure, robust, and innovative blockchain solutions.

**Our Commitment:**

- **Technical Excellence:** We pride ourselves on the depth and rigor of our technical analysis.
- **Actionable Insights:** Our reports are designed to be clear, concise, and provide practical recommendations.
- **Collaborative Partnership:** We work closely with our clients, fostering open communication throughout the audit and development lifecycle.
- **Upholding Security Standards:** We are committed to contributing to the overall security and integrity of the blockchain space.

For more information about Cashiva Community LLC, our services, and our contributions to the DeFi and blockchain ecosystem, please visit www.cashiva.com or contact us at crypto@cashiva.com.

# 4. Project Overview

## 4.1. Stated Purpose

According to project documentation provided by LanTal Mining, the LMAU is intended to bridge physical gold and blockchain technology.

*LMAu is a revolutionary digital token meticulously designed to bridge the traditional stability of physical gold with the transparency, efficiency, and global accessibility of blockchain technology. Each LMAu token represents a direct legal title to one gram of fine physical gold (99.99% purity), stored securely in fully insured, high-security vaults managed by world-class custodians.*

**IMPORTANT NOTE ON AUDIT SCOPE:**

This audit focuses **exclusively** on the technical implementation and security of the Solidity smart contract code, which manages the token on the blockchain.

The audit **DOES NOT** include verification of, and provides **NO** opinion on:

- The existence, quantity, quality, or audits of the physical gold reserves.
- The security, insurance, or procedures of the vaults or off-chain custodians.
- he validity, enforceability, or mechanism of the "direct legal title" allegedly represented by the token.
- Any off-chain minting/burning authorisation processes, attestations, or reserve management procedures.

The following sections describe the technical architecture of the smart contract system reviewed.

## 4.2. Technical System Architecture

The LMAU on-chain component is designed as an upgradeable ERC20 token. It inherits functionalities from several OpenZeppelin Upgradeable contracts:

- `Initializable`: For upgradeable contract initialization.
- `ContextUpgradeable`: Provides `_msgSender()` and `_msgData()`.
- `Ownable2StepUpgradeable`: Manages ownership with a two-step transfer process.
- `ERC20Upgradeable`: Standard ERC20 token implementation.
- `PausableUpgradeable`: Allows pausing/unpausing critical operations.
- `ERC20PermitUpgradeable`: Implements EIP-2612 for gas-less approvals.
- `UUPSUpgradeable`: For UUPS proxy pattern upgradeability.

It also introduces two custom abstract contracts:

- `Freezable`: Allows an owner to freeze and unfreeze token transfers for specific accounts.
- `Feeable`: Implements a fee-on-transfer mechanism, where a percentage of transferred tokens can be sent to a designated fee recipient.

## 4.3. Key Components

- **`LanTalMiningAuToken.sol`**: The main token contract, inheriting all functionalities.
- **`Freezable.sol`**: Abstract contract for account freezing logic using ERC-7201 custom storage.

- **`Feeable.sol`**: Abstract contract for fee-on-transfer logic using ERC-7201 custom storage. It also contains an override for the `_update` function of `ERC20Upgradeable`.

## 5. Audit Scope & Methodology

**Commit Hash:** *84e56aa2b3cf87e95d823ef59f7f1d28b1e9d08a*

- **Files in Scope:**
  - `LanTalMiningAuToken.sol` (including inherited `Freezable.sol` and `Feeable.sol`)
- **Focus Areas:**
  - Identification of common smart contract vulnerabilities within the on-chain code.
  - Correctness of upgradeability mechanisms (UUPS, storage layout).
  - Adherence to ERC20 and ERC20Permit standards.
  - Logic errors in on-chain custom features (Freezing, Fees).
  - Potential gas optimization opportunities.
  - Consistency with Solidity best practices for the smart contract code.
- **Out of Scope:**
  - Verification of off-chain reserves, legal titles, custodian processes, and attestations.
  - Deployment scripts and process.
  - Off-chain components or client-side interactions.
  - Economic model and viability.

### 5.1. Methodology

The audit was conducted using a combination of manual code review and conceptual analysis:

- **Understanding the Codebase:** Initial review to understand the architecture, intended on-chain functionality, and control flow.
- **Systematic Manual Review:** Line-by-line examination of the smart contracts to identify potential vulnerabilities based on known attack vectors and best practices.
- **Vulnerability Analysis:** Cross-referencing potential issues with common vulnerability checklists (e.g., SWC Registry).
- **Business Logic Review:** Ensuring the implemented on-chain logic aligns with the inferred intentions of the custom modules (`Freezable`, `Feeable`).
- **Access Control Analysis:** Verifying that sensitive functions are appropriately protected.
- **Upgradeability Review:** Assessing the UUPS implementation and custom storage slot management.
- **Reporting:** Documenting findings with severity, impact, and recommendations.

### 5.2. Severity Level Definitions

**Critical:** Vulnerabilities that could lead to a loss of funds, data manipulation, contract unavailability, or a takeover of contract ownership. Also includes flaws that make the contract highly unstable or prone to severe malfunction during routine maintenance, such as dependency upgrades.

**High:** Vulnerabilities that could lead to unexpected behavior, minor fund loss, or significantly hinder contract functionality, but are not as easily exploitable as Critical. (Note: No High severity findings were identified in this illustrative report, but the definition is retained for completeness).

**Medium:** Vulnerabilities that represent a deviation from best practices, could lead to inefficiencies, or have a security impact. The **likelihood** of exploitation might be lower for some medium issues, but the **potential impact** could still be significant to severe, warranting prioritized attention.

**Low:** Minor issues, such as gas optimizations or code style suggestions, that do not pose a direct security threat.

**Informational:** Observations, suggestions for code clarity, or comments that do not directly impact security but could improve maintainability or understanding.

**Resolved:** The finding has been addressed by the client.

**Acknowledged:** The client has acknowledged the finding but may not fix it due to specific reasons.

**Unresolved:** The finding has not yet been addressed.

# 6. Findings

## 6.1. Critical Severity Findings

No Critical severity findings were identified during this audit.

## 6.2. High Severity Findings

No High severity findings were identified during this audit.

## 6.3. Medium Severity Findings

### 6.3.1. LMAU-M-001: Direct ERC20 Storage Manipulation in `Feeable` Contract with Potential for Severe Impact

**Severity:** Medium

**Status:** Acknowledged

**Justification for Prioritized Attention:**

This issue is classified as Medium because it is not an immediate external exploit. The trigger event – an incompatible storage layout change in a future OpenZeppelin dependency upgrade – is a plausible scenario during routine contract maintenance. The flaw makes the contract dangerously unstable against such changes in its foundational `ERC20Upgradeable` library, with potentially catastrophic consequences for the token's ledger due to silent data corruption. It represents a significant risk to long-term data integrity and upgradeability.

**Description:**

The `Feeable` contract defines `ERC20_STORAGE_LOCATION` and uses inline assembly in `_getERC20TokenStorage()` and `_update()` to directly access and modify the parent `ERC20Upgradeable`'s storage, bypassing its intended interface.

```
// In Feeable.sol

// keccak256(abi.encode(uint256(keccak256("openzeppelin.storage.ERC20")) -
1)) & ~bytes32(uint256(0xff))
bytes32 private constant ERC20_STORAGE_LOCATION =
    0x52c63247e1f47db19d5ce0460030c497f067ca4cebf71ba98eeadabe20bace00;
```

```
// ...
function _update(address from, address to, uint256 value) internal virtual
override {
    ERC20Storage storage $erc20 = _getERC20TokenStorage();
    // ... direct manipulation of $erc20._totalSupply and $erc20._balances
}
```

**Impact:**

1. **High Likelihood of State Corruption on Dependency Upgrade:** If a future version of `ERC20Upgradeable` changes its internal storage layout, the `Feeable` contract's `_update` function will silently corrupt token balances and/or total supply.

2. **Extreme Upgrade Incompatibility & Brittleness:** The contract becomes fragile and unsafe to upgrade if its dependencies change.

3. **Violation of Encapsulation & Best Practices.**

4. **Difficulty of Recovery:** Recovering from on-chain state corruption is operationally complex and damaging.

**Recommendation:**

Refactor the `Feeable._update` function to properly use the `ERC20Upgradeable`'s internal interface.

1. Calculate the `netValue` (amount - fee) and the `feeAmount`.

2. If `netValue > 0`, call `super._update(from, to, netValue)` for the main transfer amount.

3. If `feeAmount > 0` and a valid `feeRecipient` exists, call `super._update(from, feeRecipient, feeAmount)` to transfer the fee.

Remove the `ERC20_STORAGE_LOCATION` constant and `_getERC20TokenStorage()` function entirely from `Feeable.sol`. **Addressing this is crucial for the long-term stability and safe upgradeability of the token.**

### 6.3.2. LMAU-M-002: Pausability Scope on Mint and Redeem Functions

**Severity:** Medium

**Status:** Acknowledged

**Description:**

The `onlyOwner` protected `mint()` and `redeem()` functions are not directly modified by the `whenNotPaused` modifier, which is applied to transfers and approvals.

**Impact:**

The owner can mint and redeem tokens even when the contract is paused. This may be contrary to the expectation that "pausing" halts all token supply changes and significant movements.

**Recommendation:**

Consider if this is the intended behavior. If pausing should halt **all** token operations, add the `whenNotPaused` modifier to the `mint` and `redeem` functions. If the owner must retain these capabilities during a pause, this should be clearly documented.

### 6.3.3. LMAU-M-003: Absence of an On-Chain Maximum Fee Rate Cap

**Severity:** Medium

**Status:** Acknowledged

**Description:**

The `_setFeeParams` function allows the owner to set the transfer fee rate up to 100% (`FEE_PARTS`). There is no contractually enforced, reasonable upper limit for the fee **rate**. The `maxFee_` parameter caps the absolute fee, but does not prevent a 100% rate from being applied to transfers below that cap.

**Impact:**

A malicious, compromised, or erroneous owner action could set the fee rate to an excessively high level, disrupting transfers, damaging user trust, and potentially causing loss of user funds for smaller transfers.

**Recommendation:**

Implement an on-chain, immutable `MAX_PERMISSIBLE_FEE_RATE` constant within the `Feeable` contract, representing a reasonable upper bound in percents. Modify the `_setFeeParams` function to revert if the requested `rate_` exceeds this `MAX_PERMISSIBLE_FEE_RATE`. This provides a stronger guarantee to users.

## 6.4. Low Severity Findings

No Low severity findings identified in this illustrative report based on the initial plan.

## 6.5. Informational Findings

### 6.5.1. LMAU-I-001: Visibility of Fee Calculation Logic

**Severity:** Informational

**Status:** Acknowledged

**Description:**

The `_calcFee(uint256 amount)` function within the `Feeable` abstract contract is currently marked as `internal virtual`.

```
// In Feeable.sol
function _calcFee(uint256 amount) internal virtual returns (address, uint256)
{ //... }
```

While its internal usage is correct, making this logic publicly accessible could offer benefits.

**Impact:**

Current `internal` visibility prevents external users or off-chain services from directly querying the contract to preview the fee for a potential transaction.

**Recommendation:**

Consider creating a new `public view` (or `external view`) function, say `calculateFee(uint256 amount)`, in ` LanTalMiningAuToken.sol`.

```
// In LanTalMiningAuToken.sol
function calcFee(uint256 amount) external view returns (uint256) {
    (address feeRecipient, uint256 fee) = super._calcFee(amount);
    return fee;
}
```

This enhances transparency and improves user experience for integrations.

## 7. Privileged Roles & Access Control Analysis

The `LanTalMiningAuToken` contract utilizes `Ownable2StepUpgradeable`, granting extensive and critical capabilities to a single `owner` address. This `owner` role is the sole administrative entity within the smart contract system.

**Owner Privileges (Summary):**

- **Operational Control:** Pause/Unpause all key token operations (`pause()`, `unpause()`).
- **Account Management:** Freeze/Unfreeze individual accounts (`freeze()`, `unfreeze()`), and burn all tokens from a frozen account (`burnFrozenFunds()`).
- **Token Supply Control:** Create new tokens (`mint()`), and burn tokens from the owner's balance (`redeem()`).
- **Economic Policy:** Set transfer fee rates, maximum fees, and the fee recipient (`setFeeParams()`, `setFeeRecipient()`).
- **Contract Governance:** Authorize upgrades to a new implementation logic for the entire contract (`_authorizeUpgrade()`).
- **Ownership Transfer**: Manage the transfer of the `owner` role itself (`transferOwnership()`, `acceptOwnership()`).

**Security Implications:**

The concentration of such sweeping powers into a single `owner` role means that **the security and integrity of the entire `LanTalMiningAuToken` system fundamentally depends on the security and trustworthiness of the entity controlling this single `owner` address**. If this address is compromised or acts maliciously, there are no on-chain mechanisms within the contract to prevent misuse of these powers. The subsequent "Centralization Risks & Owner Capabilities" section discusses the implications of these extensive powers in more detail.

**Recommendations:**

1. **Multisig Wallet for Owner Role (Strongly Recommended):** Replace single-signature ownership of the owner address with a multisignature (multisig) wallet (e.g., Gnosis Safe). This significantly reduces the

risk of a single point of failure by requiring multiple trusted parties to approve critical actions. A multisig setup distributes control and enhances governance security, mitigating risks from key compromise, mistakes, or malicious insiders

2. **Timelock Mechanism (Strongly Recommended):** Implement an additional timelock contract through which all `owner` actions must pass. A timelock introduces a mandatory delay between the proposal of an action and its execution. This provides transparency and a window for the community and users to review, discuss, and potentially react (e.g., by exiting positions if possible) to significant upcoming changes, providing a crucial check against immediate, unilateral actions by the `owner`.

## 8. Centralization Risks & Owner Capabilities

The design of the `LanTalMiningAuToken` contract inherently centralizes a vast amount of power in the singular `owner` role. **If the entity controlling the `owner` address is compromised, coerced, or acts maliciously, it has virtually unrestricted and unilateral ability to alter the token's operation, user balances, and fundamental logic, with no on-chain checks or balances within the contract to prevent such actions.** This section details these risks, focusing on the **capabilities** of the `owner` role as defined in the code.

### 8.1. Absolute Control Vested in the Owner Role:

The `owner` can unilaterally:

- **Manipulate Total Token Supply at Will:**
  - `mint(address to, uint256 value)`: Create an unlimited number of new tokens and assign them to any address. This can hyperinflate the supply, devaluing existing tokens.
- **Dictate Token Transferability & Economic Terms:**
  - `pause()` / `unpause()`: Instantly halt or resume all key token operations for all users.
  - `setFeeParams(uint256 rate, uint256 maxFee)`: Change the fee rate for transfers. As noted in LMAU-M-003, without an on-chain maximum rate, the owner can set this to levels that effectively confiscate transferred amounts (up to the `_max` absolute fee per transaction) or make transfers economically non-viable.
  - `setFeeRecipient(address feeRecipient)`: Redirect all collected fees to any chosen address.
- **Directly Interfere with User Accounts and Balances:**
  - `freeze(address account)` / `unfreeze(address account)`: Prevent specific users from transacting.
  - `burnFrozenFunds(address account)`: **Delete the entire token balance of any account that has been frozen by the owner.** This is an extremely powerful and potentially destructive capability with no on-chain recourse for the affected user.
- **Completely Rewrite Contract Logic:**
  - `_authorizeUpgrade(address newImplementation)`: Replace the existing token contract logic with entirely new code via the UUPS proxy. This new code could contain anything, including backdoors, different economic models, or logic to seize funds, all at the sole discretion of the `owner`.

### 8.2. Manifestation of Centralization Risks:

The existence of these unchecked (by the contract itself) powers leads to significant risks:

- **Single Point of Failure for Security:** Compromise of the single `owner` key grants the attacker complete and immediate control over all aspects of the token and its ecosystem.
- **Potential for Malicious Owner Action:** A rogue or compromised owner can:
  - Mint tokens to themselves and drain liquidity.
  - Set fees to 100% and redirect them.
  - Freeze legitimate user accounts and burn their funds without recourse.
  - Upgrade the contract to a malicious version.
- **Operational Error by Owner:** A mistake by the entity controlling the `owner` key when exercising these powerful functions can have widespread, irreversible negative consequences.
- **Target for Coercion/Regulatory Pressure:** The entity controlling the `owner` address becomes a central point of failure and a prime target for external pressure to misuse these powers against users or the system.
- **Undermining Trust and Decentralization:** For users who expect a high degree of decentralization and censorship resistance from a blockchain-based token, such extensive and unilateral owner powers are a fundamental concern, as the token's behavior and user assets are entirely subject to the `owner`'s discretion and security.

## 8.3. Mitigating Extreme Owner Powers:

Given the contract's design, mitigations primarily involve external controls or future contract design changes, as the current contract offers no internal checks on owner power.

- **Implement Timelocks (Essential External Control):** As stressed in Section 7, applying a timelock contract as an intermediary for all powerful owner functions is critical. This external mechanism would be the **only** way to introduce delay and scrutiny before owner actions take effect.
- **Enforce On-Chain Constraints (Future Design Iteration):**
  - **Fee Cap:** For future versions, consider implementing a hardcoded, reasonable maximum fee rate (see LMAU-M-003) within the contract itself.
  - **Minting Cap/Policy:** Future versions could incorporate stricter on-chain policies for minting.
  - **`burnFrozenFunds` Limitations:** Future versions could more narrowly define the conditions for `burnFrozenFunds` or require multiple independent on-chain conditions.
- **Separation of Powers (Future Design Iteration):**
  - Future designs could explore assigning certain powers to different roles or managing them via more decentralized on-chain governance mechanisms, rather than bundling all power under the single `owner`.
- **Extreme Transparency:**
  - Maintain extreme transparency regarding who or what entity controls the `owner` address and any off-chain processes or policies guiding its use. This does not change the on-chain power but can affect user trust.

**Conclusion for Centralization Risks:**

The `LanTalMiningAuToken` contract grants its `owner` a level of control that is effectively absolute over the token system. **The security and fairness of the entire LMAU ecosystem hinges entirely on the assumption that the single `owner` address will always be perfectly secured and act benignly.** The contract provides no on-chain mechanisms to limit these powers or protect users from their misuse. Implementing external controls like timelocks is essential, and for future iterations, redesigning the contract to include on-chain constraints and potentially distribute powers should be strongly considered to enhance user trust and reduce the profound risks associated with this level of centralization.

## 9. Conclusion

The `LanTalMiningAuToken` contract suite demonstrates a good use of OpenZeppelin's upgradeable contracts for its on-chain implementation. The audit identified several medium severity findings that require attention.

Notably, LMAU-M-001, concerning direct ERC20 storage manipulation in the `Feeable` contract, warrants prioritized action due to its potential for severe impact on data integrity during future dependency upgrades. Other medium findings relate to pausability scope (LMAU-M-002), and the absence of an on-chain maximum fee rate (LMAU-M-003).

A defining characteristic of this smart contract system is the absolute and unilateral power vested in the `owner` role. This role has complete control over the token's functionality, supply, economic parameters, and even its underlying logic via upgrades, as detailed in the "Centralization Risks & Owner Capabilities" section. Consequently, the integrity of the entire system and the security of user assets are wholly dependent on the singular `owner` address remaining secure and acting benevolently. This presents a critical centralization risk that users must be aware of.

Implementing external safeguards such as timelocks is strongly recommended as an immediate measure. For the long-term health and trustworthiness of the token, LanTal Mining should consider redesigning aspects of the contract in future iterations to incorporate on-chain constraints on owner powers and potentially distribute administrative controls.

By addressing all identified technical findings, and by transparently acknowledging and working to mitigate the profound centralization risks inherent in the current contract design, LanTal Mining can work towards enhancing the security, reliability, and trustworthiness of the `LanTalMiningAuToken` smart contracts.

## Appendix A: Files Audited

- `contracts/LanTalMiningAuToken.sol` (including its inherited `Freezable.sol`, `Feeable.sol`)