



Full Audit Report

LIRA DAO Token Security Assessment



LIRA DAO Token Security Assessment

FULL AUDIT REPORT

Security Assessment by SCRL on **Monday, June 24, 2024**

SCRL is deliver a security solution for Web3 projects by expert security researchers.



Executive Summary

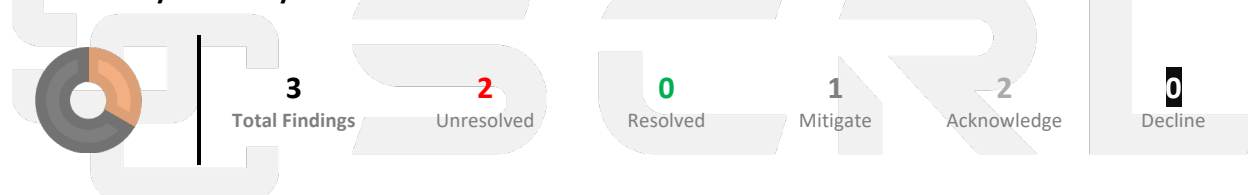
For this security assessment, SCRL received a request on Friday, May 24, 2024

Client	Language	Audit Method	Confidential	Network Chain	Contract
LIRA DAO Token	Solidity	Whitebox	Public	Arbitrum One	0x2A5E22b32b3E0Daa7a8C199e10Df9D9E1264Fd3f
Report Version	Twitter	Discord	Website		
1.1	https://twitter.com/LIRA_DAO	https://discord.com/invite/fDRBajCB9V	https://liradao.org/		

Scoring:



Vulnerability Summary



▪ 0 Critical

▪ 0 High

▪ 1 Medium 1 Mitigate

▪ 0 Low

▪ 0 Very Low

▪ 2 Informational 2 Unresolved

▪ 0 Gas-optimization

Critical severity is assigned to security vulnerabilities that pose a severe threat to the smart contract and the entire blockchain ecosystem.

High-severity issues should be addressed quickly to reduce the risk of exploitation and protect users' funds and data.

It's essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract.

While low-severity issues can be less urgent, it's still advisable to address them to improve the overall security posture of the smart contract.

Very Low severity is used for minor security concerns that have minimal impact and are generally of low risk.

Used to categorize security findings that do not pose a direct security threat to the smart contract or its users. Instead, these findings provide additional information, recommendations

Suggestions for more efficient algorithms or improvements in gas usage, even if the current code is already secure.

Audit Scope:

File	SHA-1 Hash
src/LTD.sol	5c1864b3686f04a3039569b4519bd81837153722

Audit Version History:

Version	Date	Description
1.0	Saturday, May 25, 2024	Preliminary Report
1.1	Monday, June 24, 2024	Full Audit Report

Audit information:

Request Date	Audit Date	Re-assessment Date
Friday, May 24 2024	Saturday, May 25, 2024	Monday, June 24, 2024

Smart Contract Audit Summary



**SCRL has assessed
the security of this smart contract.**

**The results of the security
assessment revealed**

No Critical Vulnerabilities.

Full Audit Report by SCRL on June 24, 2024



Security Assessment Author

Auditor:	Mark K. Kevin N. Yusheng T.	[Security Researcher Redteam] [Security Researcher Web3 Dev] [Security Researcher Incident Response]
Document Approval:	Ronny C. Chinnakit J.	CTO & Head of Security Researcher CEO & Founder

Digital Sign

Disclaimer

Regarding this security assessment, there are no guarantees about the security of the program instruction received from the client is hereinafter referred to as **"Source code"**.

And **SCRL** hereinafter referred to as **"Service Provider"**, the **Service Provider** will not be held liable for any legal liability arising from errors in the security assessment. The responsibility will be the responsibility of the **Client**, hereinafter referred to as **"Service User"** and the

Service User agrees not to be held liable to the **service provider** in any case. By contract

Service Provider to conduct security assessments with integrity with professional ethics, and transparency to deliver security assessments to users The **Service Provider** has the right to postpone the delivery of the security assessment. If the security assessment is delayed whether caused by any reason and is not responsible for any delayed security assessments.

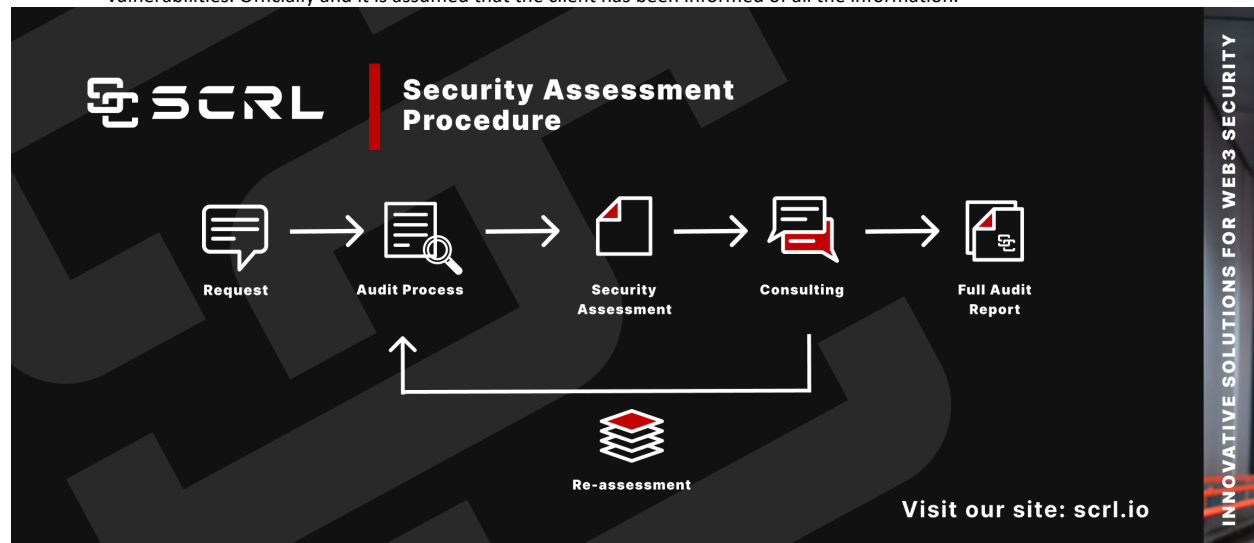
If the **service provider** finds a vulnerability The **service provider** will notify the **service user** via the Preliminary Report, which will be kept confidential for security. The **service provider** disclaims responsibility in the event of any attacks occurring whether before conducting a security assessment. Or happened later All responsibility shall be sole with the **service user**.

Security Assessment Is Not Financial/Investment Advice Any loss arising from any investment in any project is the responsibility of the investor.

SCRL disclaims any liability incurred. Whether it's Rugpull, Abandonment, Soft Rugpull, Exploit, Exit Scam.

Security Assessment Procedure

1. **Request** The client must submit a formal request and follow the procedure. By submitting the source code and agreeing to the terms of service.
2. **Audit Process** Check for vulnerabilities and vulnerabilities from source code obtained by experts using formal verification methods, including using powerful tools such as Static Analysis, SWC Registry, Dynamic Security Analysis, Automated Security Tools, CWE, Syntax & Parameter Check with AI ,WAS (Warning Avoidance System a python script tools powered by SCRL).
3. **Security Assessment** Deliver Preliminary Security Assessment to clients to acknowledge the risks and vulnerabilities.
4. **Consulting** Discuss on risks and vulnerabilities encountered by clients to apply to their source code to mitigate risks.
 - a. **Re-assessment** Reassess the security when the client implements the source code improvements and if the client is satisfied with the results of the audit. We will proceed to the next step.
5. **Full Audit Report** SCRL provides clients with official security assessment reports informing them of risks and vulnerabilities. Officially and it is assumed that the client has been informed of all the information.



Risk Rating

Risk rating using this commonly defined: $Risk\ rating = impact * confidence$

Impact The severity and potential impact of an attacker attack
Confidence Ensuring that attackers expose and use this vulnerability

Confidence	Low	Medium	High
Impact [Likelihood]			
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

Severity is a risk assessment It is calculated from the Impact and Confidence values using the following calculation methods,

$Risk\ rating = impact * confidence$

It is categorized into

7 categories severity based



For **Informational & Non-class/Optimization/Best-practices** will not be counted as severity

Category

Centralization Centralization Risk is The risk incurred by a sole proprietor, such as the Owner being able to change something without permission	Economics Risk Risks that may affect the economic mechanism system, such as the ability to increase Mint token	Logical Issue Logical Issue is that can cause errors to core processing, such as any prior operations that cause background processes to crash.	Authorization Authorization is Possible pitfalls from weak coding allows unrelated people to take any action to modify the values.	Mathematical Mathematical Any erroneous arithmetic operations affect the operation of the system or lead to erroneous values.	Naming Conventions Naming Conventions naming variables that may affect code understanding or naming inconsistencies
Security Risk Security Risk of loss or damage if it's no mitigate	Coding Style Coding Style is Tips coding for efficiency performance	Best Practices Best Practices is suggestions for improvement	Optimization Optimization is performance improvement	Gas Optimization Gas Optimization is increase performance to avoid expensive gas	Dead Code Dead Code having unused code This may result in wasted resources and gas fees.

Table Of Content

Summary

- Executive Summary
- CVSS Scoring
- Vulnerability Summary
- Audit Scope
- Audit Version History
- Audit Information
- Smart Contract Audit Summary
- Security Assessment Author
- Digital Sign
- Disclaimer
- Security Assessment Procedure
- Risk Rating
- Category

Source Code Detail

- Dependencies / External Imports
- Visibility, Mutability, Modifier function testing

Vulnerability Finding

- Vulnerability
- SWC Findings
- Contract Description
- Inheritance Relational Graph
- UML Diagram

About SCRL

Source Units in Scope

Source Units Analyzed: 1

Source Units in Scope: 1 (100%)

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	src/LTD.sol	1		28	28	17	7	16	
	Totals	1		28	28	17	7	16	

Legend: []

- **Lines:** total lines of the source unit
- **nLines:** normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC:** normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines:** lines containing single or block comments
- **Complexity Score:** a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)



Visibility, Mutability, Modifier function testing

Components


 Contracts	 Libraries	 Interfaces	 Abstract
1	0	0	0

Exposed Functions












This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

 Public	 Payable			
1	0			
External	Internal	Private	Pure	View
0	2	0	0	0

StateVariables

Total	 Public
0	0

Capabilities

Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts	
<code>^0.8.0</code>					
 Transfers ETH	 Low-Level Calls	 DelegateC all	 Uses Hash Functions	 ECRecov er	 New/Create/C reate2
 TryCatch	Σ Unchecked				

Dependencies / External Imports

Dependency / Import Path	Count
@openzeppelin/contracts/access/Ownable.sol	1
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	1

Vulnerability Findings

ID	Vulnerability Detail	Severity	Category	Status
CEN-01	Centralization Risk	Medium	Centralization	Mitigate
SEC-01	Use Ownable2Step for Secure Ownership Transfer	Informational	Best Practices	Acknowledge
SEC-02	Consider Making Constructor Payable	Informational	Best Practices	Acknowledge

CEN-01: Centralization Risk

Vulnerability Detail	Severity	Location	Category	Status
Centralization Risk	Medium	Check on finding	Centralization	Mitigate

Finding:

File: LTD.sol

```
14: contract LDT is ERC20('LIRA Dao Token', 'LDT'), Ownable {  
  
17:     constructor(address vault, address team, address marketing, address liquidity,  
address presale) Ownable(msgSender()) {  
  
    ...
```

Description:

The contract has an owner with privileged rights to mint tokens and distribute them to specific addresses (**vault, team, marketing, liquidity, presale**). This centralization introduces a risk where the owner or designated wallets could potentially dump large amounts of tokens into the market, causing significant price volatility. The lack of a token lock mechanism exacerbates this risk.

Impact:

The owner or any of the designated addresses holding a large number of tokens can sell off these tokens in the market, leading to a sharp decline in token value. This can result in loss of investor trust and financial damage to token holders. The centralization of token control without proper lock mechanisms poses a significant risk to the project's integrity.

Recommendation:

Implement a token vesting contract to lock the tokens for a specified period. This ensures that the tokens cannot be dumped into the market immediately.

Or Introduce a timelock mechanism for critical administrative functions and token transfers. This provides a delay period before changes take effect, allowing for community review and intervention if necessary.

Alleviation:

Lira DAO Team has mitigate this issue by adding

TokenDistributor - 0x70520d9BF8FE4E9eE4aCEaE6168B629961AF0A11

RewardSplitter - 0xbBBbE9b62Cab1852461D4137b10E959F5577e5BE



SEC-01: Use Ownable2Step for Secure Ownership Transfer

Vulnerability Detail	Severity	Location	Category	Status
Use Ownable2Step for Secure Ownership Transfer	Informational	Check on finding	Best Practices	Acknowledge

Finding:

```
14| contract LDT is ERC20('LIRA Dao Token', 'LDT'), Ownable {
```

The contract currently inherits from the Ownable contract, which allows the transfer of ownership to a new address in a single transaction. This can potentially lead to accidental transfers of ownership to an address that may not be able to handle the responsibilities, such as a non-contract address or an address with no known owner.

By demanding that the receiver of the owner permissions actively accept via a contract call of its own, Ownable2Step prevents the contract ownership from accidentally being transferred to an address that cannot handle it. This makes the ownership transfer process more secure by ensuring that the new owner explicitly accepts the ownership.

Recommendation:

Replace the usage of Ownable with Ownable2Step to ensure a secure two-step ownership transfer process. This change ensures that the new owner has to explicitly accept the ownership, reducing the risk of accidental or malicious ownership transfers.

Reference: Ownable2Step documentation
<https://docs.openzeppelin.com/contracts/4.x/api/access#Ownable2Step>

Alleviation:

-

SEC-02: Consider Making Constructor Payable

Vulnerability Detail	Severity	Location	Category	Status
Consider Making Constructor Payable	Informational	Check on finding	Best Practices	Acknowledge

Finding:

```
17|     constructor(address vault, address team, address marketing, address liquidity,  
18|         address presale) Ownable(_msgSender()) {  
19|         _mint(vault, 9_000_000_000 * 10 ** 18);  
20|         _mint(team, 50_000_000 * 10 ** 18);  
21|         _mint(marketing, 100_000_000 * 10 ** 18);  
22|         _mint(liquidity, 100_000_000 * 10 ** 18);  
23|         _mint(presale, 750_000_000 * 10 ** 18);  
24|     }
```

The constructor in the current implementation is non-payable. Making the constructor payable can save gas, as the contract creation transaction may be sent with a small amount of Ether (even zero), allowing for a more optimized gas usage. This change is particularly beneficial if the contract deployment will be done with some Ether attached, which is common in many deployment scenarios.

Recommendation:

Update the constructor to be payable. This will allow the contract to receive Ether during deployment, which can optimize the gas usage

Reference: Constructor documentation
<https://docs.soliditylang.org/en/v0.8.0/contracts.html#constructors>

Alleviation:

-



SWC Findings

ID	Title	Scanning	Result
SWC-100	Function Default Visibility	Complete	No risk
SWC-101	Integer Overflow and Underflow	Complete	No risk
SWC-102	Outdated Compiler Version	Complete	No risk
SWC-103	Floating Pragma	Complete	No risk
SWC-104	Unchecked Call Return Value	Complete	No risk
SWC-105	Unprotected Ether Withdrawal	Complete	No risk
SWC-106	Unprotected SELFDESTRUCT Instruction	Complete	No risk
SWC-107	Reentrancy	Complete	No risk
SWC-108	State Variable Default Visibility	Complete	No risk
SWC-109	Uninitialized Storage Pointer	Complete	No risk
SWC-110	Assert Violation	Complete	No risk
SWC-111	Use of Deprecated Solidity Functions	Complete	No risk
SWC-112	Delegatecall to Untrusted Callee	Complete	No risk
SWC-113	DoS with Failed Call	Complete	No risk
SWC-114	Transaction Order Dependence	Complete	No risk
SWC-115	Authorization through tx.origin	Complete	No risk



SWC-116	Block values as a proxy for time	Complete	No risk
SWC-117	Signature Malleability	Complete	No risk
SWC-118	Incorrect Constructor Name	Complete	No risk
SWC-119	Shadowing State Variables	Complete	No risk
SWC-120	Weak Sources of Randomness from Chain Attributes	Complete	No risk
SWC-121	Missing Protection against Signature Replay Attacks	Complete	No risk
SWC-122	Lack of Proper Signature Verification	Complete	No risk
SWC-123	Requirement Violation	Complete	No risk
SWC-124	Write to Arbitrary Storage Location	Complete	No risk
SWC-125	Incorrect Inheritance Order	Complete	No risk
SWC-126	Insufficient Gas Griefing	Complete	No risk
SWC-127	Arbitrary Jump with Function Type Variable	Complete	No risk
SWC-128	DoS With Block Gas Limit	Complete	No risk
SWC-129	Typographical Error	Complete	No risk
SWC-130	Right-To-Left-Override control character (U+202E)	Complete	No risk
SWC-131	Presence of unused variables	Complete	No risk
SWC-132	Unexpected Ether balance	Complete	No risk

SWC-133	Hash Collisions With Multiple Variable Length Arguments	Complete	No risk
SWC-134	Message call with hardcoded gas amount	Complete	No risk
SWC-135	Code With No Effects	Complete	No risk
SWC-136	Unencrypted Private Data On-Chain	Complete	No risk

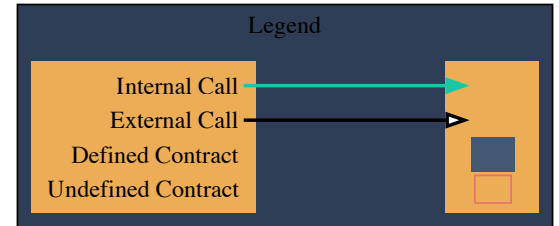
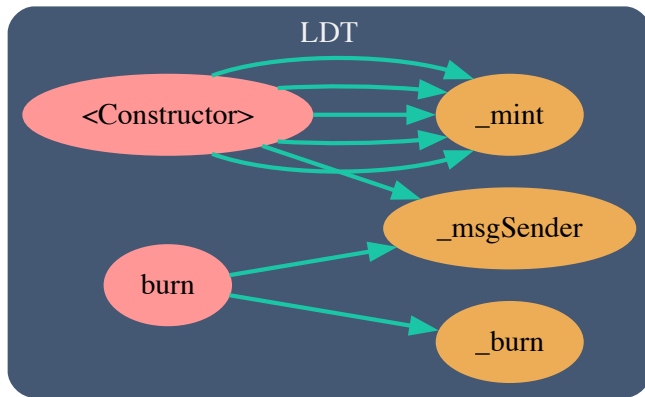
Contracts Description Table

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
LDT	Implementation	ERC20, Ownable		
L		Public !		Ownable
L	burn	Public !		NO !

Legend

Symbol	Meaning
	Function can modify state
	Function is payable

Call Graph



UML Class Diagram

LDT LTD.sol
Public: constructor(vault: address, team: address, marketing: address, liquidity: address, presale: address) burn(quantity: uint256)

About SCRL

SCRL (Previously name SECURI LAB) was established in 2020, and its goal is to deliver a security solution for Web3 projects by expert security researchers. To verify the security of smart contracts, they have developed internal tools and KYC solutions for Web3 projects using industry-standard technology. SCRL was created to solve security problems for Web3 projects. They focus on technology for conciseness in security auditing. They have developed Python-based tools for their internal use called WAS and SCRL. Their goal is to drive the crypto industry in Thailand to grow with security protection technology.



Support ALL EVM L1 - L2

Smart Contract Audit

Our top-tier security strategy combines static analysis, fuzzing, and a custom detector for maximum efficiency.

scrl.io



Follow Us On:

Website	https://scrl.io/
Twitter	https://twitter.com/scrl_io
Telegram	https://t.me/scrl_io
Medium	https://scrl.medium.com/