QuillAudits

AUDIT REPORT

February 2026

For

LCX

Table of Content

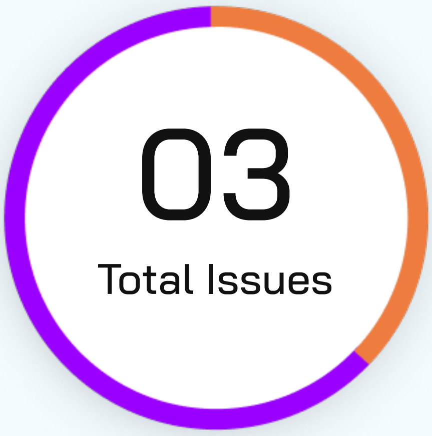
Executive Summary

Project Name	LCX
Protocol Type	Token Contract
Project URL	https://exchange.LCX.com/
Overview	LCX Token is an upgradeable ERC20 token designed for the LCX ecosystem. Features: - Upgradeable architecture using TransparentUpgradeableProxy - Account blacklisting with recovery mechanism - Global pause functionality for emergency controls - Cross-chain compatibility with OP Stack's L1StandardBridge
Audit Scope	The scope of this Audit was to analyze the LCX Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/LCX-AG/LCX-Token-Contract-Upgraded/tree/main/src/token
Contracts in Scope	src/token/Token.sol src/token/TokenAdmin.sol
Branch	Main
Commit Hash	017c949ee68cd06bd4ef34ee316904d718da436e
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	2nd February 2026
Updated Code Received	3rd February 2026
Review 2	3rd February 2026
Fixed In	0cde3a5bf836efc9b18eb4396d89cd078f3bb8d7

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	1 (33.3%)
🟨	Low	0(0.0%)
🟪	Informational	2 (66.6%)

Severity

Issues

	Critical	High	Medium	Low	Informational
Open	0	0	0	0	0
Acknowledged	0	0	**1**	0	0
Partially Resolved	0	0	0	0	0
Resolved	0	0	0	0	**2**

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Blacklist Enforcement Can Be Front-Run Before Confirmation	Medium	Acknowledged
2	Missing Initializer Call for `PausableUpgradeable`	Informational	Resolved
3	Missing Zero Address Validation in One-Time Initialization Functions	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

Likelihood \ Impact	🟥 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.

Medium Severity Issues

Blacklist Enforcement Can Be Front-Run Before Confirmation

Acknowledged

Path
src/TokenAdmin.sol

Function Name
`blacklist(address)`

Description
The `blacklist` mechanism can be front-run by a targeted user. Since blacklisting is executed via an on-chain transaction, the account being blacklisted can observe the pending transaction in the mempool and preemptively transfer or burn their tokens before the blacklist is finalized. This is for chains with public mempool.

Impact
Blacklisted users may retain control of their funds by moving them to non-blacklisted addresses, defeating the purpose of the blacklist mechanism. This undermines enforcement actions, compliance requirements, and any assumptions that blacklisting can reliably freeze assets.

Likelihood
Medium. Front-running via mempool monitoring is trivial and commonly used, especially in adversarial environments. Any user with basic blockchain knowledge can exploit this behavior.

POC
1. Blacklister submits a `blacklist(account)` transaction.
2. The target `account` observes the pending transaction in the mempool.
3. Before the blacklist transaction is mined, `account` calls:

```solidity
transfer(attackerControlledAddress, balance);
```

4. Tokens are successfully transferred since `whenNotBlacklisted(_msgSender())` still passes.
5. The blacklist transaction is mined, but the account now holds no tokens.

Recommendation
Consider operational mitigations such as submitting blacklist transactions via a private transaction relay to reduce mempool visibility prior to confirmation.

Informational Issues

Missing Initializer Call for `PausableUpgradeable`

Resolved

Path

src/TokenAdmin.sol

Function

`__LCXTokenAdmin_init)`

Description

The contract inherits from `PausableUpgradeable` but does not invoke `__Pausable_init()` during initialization.

`LCXToken` inherits `PausableUpgradeable` via `LCXTokenAdmin`. However, the initializer does not call `__Pausable_init()`. While the paused state defaults to `false` and the contract functions correctly at runtime, OpenZeppelin upgradeable contracts require that all inherited modules be explicitly initialized.

Failing to do so breaks standard initialization patterns and may introduce upgrade safety issues in future contract versions.

Recommendation

Invoke `__Pausable_init()` in the contract initializer.

```solidity
__Pausable_init();
```

Missing Zero Address Validation in One-Time Initialization Functions

Resolved

Path

src/TokenAdmin.sol

Function

`initMigrator(address lcxMigrator)`

`initRemoteToken(address l2Token)`

`initOpBridge(address l1StandardBridge)`

Description

The contract's one-time initialization functions do not validate that the provided addresses are non-zero. Also these values cannot be updated after being set and will require redeploying a new proxy entirely.

Recommendation

Add explicit zero-address validation to all initialization functions.

```solidity
require(lcxMigrator != address(0), "LCXToken: zero address");
require(l2Token != address(0), "LCXToken: zero address");
require(l1StandardBridge != address(0), "LCXToken: zero address");
```

Functional Tests

Some of the tests performed are mentioned below:

- ✓ Should be able to transfer and transfer from when not paused
- ✓ Blacklisted address should not be able to transfer or receive tokens
- ✓ Admin/Blacklister should be able to blacklist accounts
- ✓ Only admin should be able to de-blacklist an account
- ✓ Only bridge should be able to burn tokens
- ✓ Only migrator and bridge should be able to mint tokens
- ✓ Blacklisted address should only be able to transfer tokens back the owner

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

1. External Dependencies & Trust Boundaries

Dependency	Type	How It Is Used	Trust Assumption	Associated Risks
OpenZeppelin ERC20Upgradeable	Library	Base ERC20 functionality	Correct implementation, no vulnerabilities	Token transfer logic bugs, approval race conditions
OpenZeppelin Ownable2StepUpgradeable	Library	Two-step ownership transfer	Correct implementation	Ownership hijacking if pending owner compromised
OpenZeppelin PausableUpgradeable	Library	Emergency pause mechanism	Correct implementation	Ownership hijacking if pending owner compromised
OpenZeppelin Ownable2Step (v5)	Library	ProxyAdmin ownership	Correct implementation	Proxy upgrade authority compromise
OpenZeppelin ERC1967Proxy	Library	Proxy implementation	Correct implementation	Storage collision, delegatecall vulnerabilities
OpenZeppelin ERC1967Utils	Library	Proxy upgrade mechanics	Correct implementation	Implementation slot manipulation
(IERC20Burnable)	Contract	for migration (burn)	implementation, correct burn behavior	reentrancy, accounting mismatch
OP Stack L1StandardBridge	External Contract	Cross-chain mint/burn authority	Bridge is not compromised	Unlimited minting if bridge compromised

Dependency	Type	How It Is Used	Trust Assumption	Associated Risks
L2 Remote Token	External Contract	Bridge counterpart on L2	Correct pairing with L1 token	Cross-chain desync, incorrect peg
Proxy Implementation Slot	Storage	ERC1967 implementation address	Not externally writable	Implementation slot overwrite attack

2. Asset Flow Mapping

2.1 Asset-Holding Contracts

Contract	Asset Type	Custodied Assets	Notes
LCXToken (Proxy)	ERC20 (LCX)	User balances (token ledger)	Primary token contract - balances are accounting entries

2.2 Asset Entry & Exit Functions

Contract	Function	Asset In	Asset Out	Caller	Risk Notes
LCXToken	mint (migrator)	-	LCX (new)	Migrator	Mints backed by burned old tokens (external enforcement)

Contract	Function	Asset In	Asset Out	Caller	Risk Notes
LCXToken	mint (bridge)	-	LCX (new)	OP Bridge	Mints backed by L2 burns; NO RESTRICTIONS applied
LCXToken	burn	-	LCX (new)	Migrator	Burns for L2 bridging; requires allowance if caller != from
LCXToken	transfer	-	-	Any	Standard transfer; blacklist/pause checked
LCXToken	transferFrom	-	-	Any	Approved transfer; blacklist/pause checked
LCXToken	recoverBlacklistedBalance	-	LCX (full balance)	Blacklisted user	Full balance to owner; effective fund seizure
LCXToken	recoverLockedTokens	-	LCX (contract balance)	Owner	Recovery of stuck tokens

3. Trust & Privilege Model

3.1 Privileged Roles

Role	Contract	Powers	Trust Level	Risk
Owner (Token)	LCXToken	Pause/unpause, assign/revoke blacklisters, de-blacklist, recover tokens, set migrator/bridge remoteToken (one-time)	CRITICAL	Complete protocol control; can freeze all operations, seize blacklisted funds
Owner (Migrator)	LCXToken Migrator	Pause/unpause migrations, recover locked tokens	HIGH	Can halt migrations permanently
Owner (ProxyAdmin)	ProxyAdmin	Upgrade proxy implementation	CRITICAL	Can replace entire token logic; complete takeover possible
Blacklister	LCXToken Admin	Blacklist any address	HIGH	Can freeze arbitrary accounts; potential censorship/griefing
Migrator Contract	LCXToken	Mint tokens (with pause/blacklist checks)	HIGH	Unlimited supply inflation if compromised
OP Bridge Contract	LCXToken	Mint tokens (NO restrictions), burn tokens	HIGH	Unlimited unrestricted supply inflation if compromised

Closing Summary

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

No critical issues in LCX token, just 1 issue of medium severity and 2 informational severity was found. LCX team fixed two and acknowledged the remaining issue

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

8+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1500+** Projects Secured

Follow Our Journey

AUDIT REPORT

February 2026

For

LCX

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com