Smart Contract Security & Blockchain Development

LF Wallet Token (LWT)

Final Security Audit Report

Report Type: Final Audit Report • Version 1.0 • Deployed Contract Review

Field	Detail
Project	LF Wallet Token — fixed-supply BEP-20 utility token with EIP-2612 permit (gasless approvals)
Contract in Scope	`LFWalletToken` (inherits OpenZeppelin `ERC20` · `ERC20Permit` · `EIP712` · `Nonces`) + 20 OpenZeppelin 5.7.0 library files
Contract Address	`0x0c9Eaa616CDdD5b0D94325635b5c611210E513d6` (BscScan: Source Code Verified — Exact Match)
Report Type	Final Audit Report — v1.0
Prepared By	EtherAuthority Security Team
Approved By	Yogesh P. (Lead Security Engineer)
Audit Date	October 31, 2026
Network	BNB Smart Chain (Mainnet, chain id 56)
Language / Compiler	Solidity ^0.8.27 (deployed with v0.8.27+commit.40a35a09, optimizer disabled, default EVM version), OpenZeppelin Contracts 5.7.0
Website	https://lfwallet.app

CONFIDENTIAL - Prepared exclusively for the client. Not for public distribution without written consent.

Table of Contents

Document Control	3
Executive Summary	4
Project Overview	6
Audit Scope	6
Methodology	8
Critical / High / Medium Findings	11
Low Findings	11
Informational Findings	13
Gas Optimizations	15
Centralization & Privileged Roles	17
Deployment Verification	17
Final Verdict	19
Disclaimer	19

Document Control

Report Version History

Version	Date	Report Type	Description
v1.0	Oct 31, 2026	Final Audit Report	This report. Review of the verified source code of the contract deployed at `0x0c9E…13d6` on BNB Smart Chain, including manual review, Slither static analysis, 33 executed dynamic tests on a local EVM, and a byte-for-byte comparison of the deployed runtime bytecode against the reviewed source. Because the contract is already deployed and is not upgradeable, no remediation cycle applies; every finding carries its final status.

Audit Report Lifecycle

EtherAuthority follows the same staged process used by leading audit firms. For a contract that is reviewed after deployment and cannot be changed, the Preliminary and Final stages collapse into a single Final Audit Report, as shown below.

Stage	Document	What happens
1. Code & deployment review	Internal working notes	Full review of the verified source, static analysis, executed tests, and verification that the on-chain runtime bytecode is produced by the reviewed source.
2. Client review	Client response per finding	The client reviews each finding. For an immutable contract the possible responses are ACKNOWLEDGED (accepted as is, documented for users) or a change in a future deployment.
3. Final Audit Report	Final Audit Report — v1.0 (this document)	Findings carry their final status: RESOLVED, MITIGATED, ACKNOWLEDGED, NO ACTION NEEDED or OPEN. The Final Verdict is issued at this stage. **(Current stage)**

Executive Summary

Scope: the single application contract `LFWalletToken` (16 lines) deployed at `0x0c9Eaa616CDdD5b0D94325635b5c611210E513d6` on BNB Smart Chain, together with the 20 OpenZeppelin Contracts 5.7.0 library files it inherits from (4,696 lines in total, all verified on BscScan as an exact match). The contract is a plain, fixed-supply BEP-20 token: it mints 30,000,000,000 LWT (18 decimals) to one recipient in its constructor and adds EIP-2612 `permit` for gasless approvals. It has no owner, no admin functions, no mint, burn, pause, fee, tax, blacklist or upgrade mechanism, so its behaviour is fixed for ever.

Methodology: three independent manual review passes over the application contract and the inherited OpenZeppelin code paths (ERC-20 transfer/approval logic, EIP-712 domain handling, ECDSA signature recovery and nonce management) were combined with Slither 0.11 static analysis (102 detectors, 75 raw results, all triaged by hand — none is a vulnerability in the application contract), 33 executed dynamic tests on a local EVM (py-evm, Cancun rules) covering deployment, transfers, approvals, permit signatures, replay and malleability attacks, cross-chain replay and a 300-step random-transfer fuzz, and a byte-for-byte comparison of the deployed runtime bytecode (6,802 bytes) with the bytecode compiled from the reviewed source. All 33 tests passed and the bytecode matched (differences confined to the compiler's immutable values and metadata hash, see Deployment Verification).

Final Assessment: No Critical, High or Medium severity vulnerability was identified. Two Low issues were recorded: the standard ERC-20 approval race condition (L-01) and the absence of any way to recover tokens sent to the token contract itself (L-02). Five Informational items and one Gas item were also noted, the most relevant for token holders being that 100% of the supply was minted to a single externally owned account and remains there at the time of writing (I-02). Because the contract is immutable and none of the findings allows loss or theft of funds through the contract code, no remediation is required; the Low and Informational items should be documented for users and addressed in any future deployment.

Severity Count Summary



Severity	Count	Share
Critical	0	0.0%
High	0	0.0%
Medium	0	0.0%
Low	2	25.0%
Informational	5	62.5%
Gas Optimization	1	12.5%
Total findings	**8**	**100%**

Fix status of all 8 findings: 0 Resolved ● 5 Acknowledged / Open (immutable contract — Low and Informational only) ● 3 No Action Needed (L-01, I-01, I-03)

Summary for Stakeholders

LF Wallet Token is as simple as a token contract can be: a copy of the industry-standard OpenZeppelin ERC-20 implementation with the widely used permit extension and a one-line constructor that creates the whole supply. We found nothing in the code that would let anyone — including the deployer — take tokens from holders, block transfers, create new tokens, change fees or otherwise alter the rules after deployment. There are no fees or taxes on transfers, no trading restrictions, and no administrator of any kind.

The points holders should know about are not code defects but properties of the design. The entire 30 billion LWT supply was sent to one wallet (`0x24AF…8183`, the deployer) and that wallet still holds 100% of it, so the project team controls distribution and market supply entirely off-chain. Tokens accidentally sent to the token contract's own address cannot be recovered. And as with every standard ERC-20 token, changing an existing allowance should be done by first setting it to zero (or by using `permit`) to avoid the well-known approval race.

We confirmed that the code published on BscScan is exactly what is running on chain. The contract is suitable for use as a fixed-supply token.

Project Overview

LF Wallet Token (symbol LWT) is a BEP-20 token on BNB Smart Chain intended as the utility token of the LF Wallet project. The contract was deployed on October 29, 2026 (block 130,539,122, transaction `0xc32b79e8…7b5768`) by the externally owned account `0x24AFd1BAbAf101A80a6000c2c865aE6261BA8183`, which was also passed as the constructor's `recipient`. The constructor mints 30,000,000,000 LWT ($30 \times 10^9 \times 10^{18}$ base units) to that address in a single `Transfer` event. No further tokens can ever be created and none can be destroyed except by sending them to an unreachable address.

The token exposes the thirteen functions of the ERC-20 and EIP-2612 standards. `permit` lets a holder sign an EIP-712 message off-chain so that a third party (for example a dApp or a relayer) can set an allowance on the holder's behalf and pay the gas; each signature is bound to the holder's nonce, a deadline, the contract address and chain id 56.

Contract	Role	Holds funds	Privileged roles
LFWalletToken	Fixed-supply BEP-20 token (ERC-20 + EIP-2612 permit); mints the full supply to the constructor recipient; standard transfer and allowance logic from OpenZeppelin 5.7.0	No (holds no BNB; any LWT sent to it is stuck — L-02)	None. No owner, admin, minter, pauser or upgrade authority.

Audit Scope

The audit covered the verified Solidity Standard-JSON source published on BscScan for the contract address `0x0c9Eaa616CDdD5b0D94325635b5c611210E513d6` (compiler v0.8.27+commit.40a35a09, optimizer disabled, 200 runs, default EVM version). The application contract is file #1; files #2–#21 are unmodified OpenZeppelin Contracts 5.7.0 sources, which were confirmed identical to the official npm release and reviewed for the code paths the token actually uses.

#	Source File	Lines	SHA-256 (first 16 hex)
1	contract-f61dacff88.sol (`LFWalletToken`)	16	18b05dbacde09701
2	@openzeppelin/contracts@5.7.0/token/ERC20/ERC20.sol	306	50f34ae16a067a41
3	@openzeppelin/contracts@5.7.0/token/ERC20/IERC20.sol	80	01b6f5c4fa45fd38
4	@openzeppelin/contracts@5.7.0/token/ERC20/extensions/IERC20Metadata.sol	27	0b7132f17d14d1d8
5	@openzeppelin/contracts@5.7.0/utils/Context.sol	29	847fda5460fee70f
6	@openzeppelin/contracts@5.7.0/interfaces/draft-IERC6093.sol	163	fa7068f56bc18057
7	@openzeppelin/contracts@5.7.0/token/ERC20/extensions/ERC20Permit.sol	78	53584e84cd485144
8	@openzeppelin/contracts@5.7.0/token/ERC20/extensions/IERC20Permit.sol	91	c302edfbf7f396e1
9	@openzeppelin/contracts@5.7.0/utils/cryptography/ECDSA.sol	285	e3758f08ce5e53ad
10	@openzeppelin/contracts@5.7.0/utils/cryptography/EIP712.sol	169	1023ca2a7f959076
11	@openzeppelin/contracts@5.7.0/utils/cryptography/MessageHashUtils.sol	229	538a0d7c4f0ec5a8
12	@openzeppelin/contracts@5.7.0/utils/Strings.sol	533	e318709bbe73a783

#	Source File	Lines	SHA-256 (first 16 hex)
13	@openzeppelin/contracts@5.7.0/utils/math/Math.sol	764	bdfdbe133991c0c7
14	@openzeppelin/contracts@5.7.0/utils/Panic.sol	58	270fc8401c1a13fa
15	@openzeppelin/contracts@5.7.0/utils/math/SafeCast.sol	1,163	5779bc848bde39f1
16	@openzeppelin/contracts@5.7.0/utils/math/SignedMath.sol	69	1ed50b1056af8867
17	@openzeppelin/contracts@5.7.0/utils/Bytes.sol	330	78b01e1da6f6e6aa
18	@openzeppelin/contracts@5.7.0/utils/ShortStrings.sol	123	0b66f6166641f195
19	@openzeppelin/contracts@5.7.0/utils/StorageSlot.sol	144	75704538dcb22323
20	@openzeppelin/contracts@5.7.0/interfaces/IERC5267.sol	29	f89772d2d4fafcbe
21	@openzeppelin/contracts@5.7.0/utils/Nonces.sol	48	6a98ef6eaaac8dfc

Hashes are SHA-256 of each file exactly as published in the BscScan Standard-JSON input. Files 12–19 are pulled in transitively by `EIP712`/`ECDSA` (string handling and maths) and contribute no reachable logic beyond `ShortStrings` and `Strings` usage in `eip712Domain()`.

Reviewed application source

contract-f61dacff88.sol — *the complete application contract*

```
// SPDX-License-Identifier: MIT
// Compatible with OpenZeppelin Contracts ^5.7.0
pragma solidity ^0.8.27;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import {ERC20Permit} from "@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";

contract LFWalletToken is ERC20, ERC20Permit {
    constructor(address recipient)
        ERC20("LF Wallet Token", "LWT")
        ERC20Permit("LF Wallet Token")
    {
        _mint(recipient, 30000000000 * 10 ** decimals());
    }
}
```

Methodology

The engagement followed a layered approach. No finding is reported from a scanner alone: every candidate issue was traced through the code by hand and, wherever possible, reproduced by an executed test.

Phase	Activities	Output
1. Scoping & threat model	Architecture mapping, trust boundaries, privileged roles (none), value flows, external dependencies (OpenZeppelin 5.7.0 only — no oracles, routers or external calls).	Attack surface list (13 external functions, 4 state-changing)
2. Manual code review	Three independent passes over the application contract and the inherited OpenZeppelin paths it uses: `_mint`/`_update`, `_approve`/`_spendAllowance`, `permit`, EIP-712 domain caching and `eip712Domain()`, ECDSA recovery and malleability checks, nonce handling, constructor ordering and immutables.	Candidate findings
3. Static analysis	Slither 0.11.x, 102 detectors on the full Standard-JSON compilation unit (23 contracts). 75 raw results triaged by hand: 42 assembly, 9 divide-before-multiply and 1 incorrect-exp in OpenZeppelin `Math` (known false positives), 8 too-many-digits, 4 dead-code, 4 solc-version, 4 naming-convention, 1 shadowing-local, 1 timestamp (permit deadline, intended), 1 pragma. Two map to I-01 and I-05.	0 vulnerabilities in the application contract
4. Dynamic testing	33 executed tests on a local EVM (py-evm, Cancun) against bytecode compiled with the exact deployed settings: deployment invariants and events, transfers (exact amounts, zero value, self, to the contract, to address(0), over-balance), approvals and `transferFrom` (allowance accounting, infinite allowance, zero spender), `permit` (valid relay, replay, expiry, tampering, wrong chain id, high-s malleability, garbage signature, front-run) and a 300-step random-transfer fuzz checking supply conservation.	33 / 33 passed; execution log and gas figures
5. Deployment verification	Compiled the reviewed source with solc 0.8.27 (optimizer off) and compared the 6,802-byte runtime bytecode with `eth_getCode` from a BNB Smart Chain node; decoded the constructor argument; read `totalSupply`, `decimals`, `balanceOf(recipient)` and `nonces` on chain.	Match confirmed (see Deployment Verification)
6. Exploit simulation	Attempts to re-use or forge permit signatures, replay across chains, exploit signature malleability, exceed balances or allowances, and mint or burn through any reachable path.	No exploit found
7. Reporting	Final report with statuses and verdict.	This document

Security Checks Performed

Category	Result
Reentrancy & callback safety	Not applicable — the contract makes no external calls and never transfers BNB; all state updates follow checks-effects pattern in OpenZeppelin `_update`.
Access control & privileged functions	No privileged function exists. `_mint` is reachable only from the constructor.
Arithmetic, rounding & precision	Solidity 0.8 checked arithmetic; supply 3×10^{28} is far below `uint256` limits; fuzz confirmed supply conservation.
Oracle & price manipulation	Not applicable — no price logic.
Front-running / MEV	Standard approval race (L-01); permit front-run cannot change the outcome (I-03).
Denial of service & gas griefing	No loops, no unbounded storage, no external dependencies; transfers cannot be blocked.
Signature security (EIP-712 / EIP-2612)	Domain separator bound to chain id 56 and contract address with cache invalidation on fork; nonces strictly increasing; deadline enforced; high-s signatures and invalid recoveries rejected (tests T25–T30).
Token standard compliance	ERC-20 (returns `bool`, emits `Transfer`/`Approval`), ERC-20 Metadata, EIP-2612, ERC-5267, ERC-6093 custom errors. No fee-on-transfer, no rebasing, no blacklist, no pause.
Honeypot / rug-pull patterns	None: no hidden mint, no owner-only transfer switch, no max-wallet or max-tx, no tax, no proxy, no `selfdestruct`.
Input validation	Zero-address sender, receiver, owner and spender rejected by OpenZeppelin (tests T06, T09, T18).
Locked funds / asset recovery	Tokens sent to the contract itself are unrecoverable (L-02); the contract cannot receive BNB (no `receive`/`fallback`).
Upgradeability & proxy risks	Not applicable — immutable contract, no proxy.
Centralization	No on-chain privileges; token distribution is fully concentrated (I-02) — see Centralization section.
Compiler & dependencies	solc 0.8.27 (no known bugs affecting this code); OpenZeppelin 5.7.0 unmodified, no known advisories for the used modules.
Code quality & gas	I-01, I-04, I-05, G-01

Function-Level Attack Surface

Every external entry point of the deployed contract (13 functions, including inherited ones) was listed with who can call it, what it changes and which guards protect it.

Function	State change	Guards verified	Findings

`transfer(to, value)`	Yes	Non-zero sender/receiver; balance check; checked arithmetic; event	L-02
`approve(spender, value)`	Yes	Non-zero owner/spender; event	L-01
`transferFrom(from, to, value)`	Yes	Allowance check and decrement (infinite allowance kept); then `_transfer` guards	L-01
`permit(owner, spender, value, deadline, v, r, s)`	Yes	Deadline; EIP-712 typed hash; ECDSA recover with malleability check; signer == owner; nonce consumed before approval	I-03
`name, symbol, decimals, totalSupply, balanceOf, allowance, nonces, DOMAIN_SEPARATOR, eip712Domain`	No (view)	Pure reads; domain separator recomputed if chain id or address changes	—
`constructor(recipient)`	Once	Rejects `recipient == address(0)`; single mint	I-02, I-05

Severity Classification

Severity	Definition
Critical	Direct theft or permanent loss of significant funds, or full protocol takeover; easy to exploit.
High	Loss of funds or severe disruption under realistic conditions.
Medium	Disruption of a core function or limited value loss; funds recoverable or conditions specific.
Low	Minor impact, bounded loss, or requires unlikely conditions or a trusted actor.
Informational	No direct security impact: design notes, best practice, documentation.
Gas Optimization	Reduces gas cost without changing behaviour.

Critical, High and Medium Findings

No Critical, High or Medium severity issues were identified. All three manual review passes, the static analysis and the 33 executed tests agree: there is no path through which tokens can be created, destroyed, frozen, taxed or taken from a holder other than by the holder's own transfer, approval or signed permit.

Low Findings

L-01 — Standard ERC-20 Approval Race Condition (Front-Running of `approve`)

LOW **Status: NO ACTION NEEDED (inherent to the ERC-20 standard)**

Description

When a holder changes an existing non-zero allowance with `approve`, a spender who watches the mempool can spend the old allowance before the new value is written and then spend the new one as well, receiving up to old + new tokens. This is the well-known ERC-20 approval race (test T20 confirms that the token offers no `increaseAllowance`/`decreaseAllowance` helpers, as OpenZeppelin removed them in v5). It requires a malicious, already-approved spender and bounded loss, so it is rated Low.

Relevant code — @openzeppelin/contracts/token/ERC20/ERC20.sol — *approve overwrites the allowance unconditionally*

```
120 function approve(address spender, uint256 value) public virtual returns (bool) {
121     address owner = _msgSender();
122     _approve(owner, spender, value);
123     return true;
124 }
```

Recommendation

No contract change is possible or needed; this is the behaviour of every standard ERC-20 token. Front-ends and integrators should change an allowance by first setting it to zero, or by granting exact amounts through `permit` and spending them in the same transaction. Users should approve only the amount they intend to spend.

L-02 — Tokens Sent to the Token Contract Itself Cannot Be Recovered

LOW **Status: ACKNOWLEDGED / OPEN (immutable contract)**

Description

The contract accepts LWT transfers to its own address like any other recipient and has no function to move them out again, so such tokens are lost for ever. Test T12 transferred 1 base unit to the contract and confirmed `balanceOf(token) == 1` with no path to withdraw it. This is a common user error with ERC-20 tokens (the address is shown on every explorer page). Other ERC-20 tokens accidentally sent to this contract are equally stuck; BNB cannot be sent at all because there is no `receive` function.

Relevant code — @openzeppelin/contracts/token/ERC20/ERC20.sol — *only address(0) is rejected as a receiver*

```
function _transfer(address from, address to, uint256 value) internal {
    if (from == address(0)) {
        revert ERC20InvalidSender(address(0));
    }
    if (to == address(0)) {
        revert ERC20InvalidReceiver(address(0));
    }
    _update(from, to, value);
}
```

Recommendation

Document in the project's user material that transfers to the contract address are irreversible. In any future deployment, either reject `to == address(this)` in an `_update` override or add a narrowly scoped rescue function behind a multisig that can only move tokens held by the contract itself. No change is possible to the deployed contract.

Informational Findings

I-01 — Floating Pragma `^0.8.27`

INFORMATIONAL **Status: NO ACTION NEEDED (deployed compiler verified)**

Description

The application contract uses a caret pragma, which would allow any 0.8.x compiler from 0.8.27 upward. For a deployed contract this is only a reproducibility concern: the BscScan verification fixes the compiler to v0.8.27+commit.40a35a09, and our recompilation with that exact version reproduced the on-chain runtime bytecode. No known solc bug in 0.8.27 affects this code.

Relevant code — contract-f61dacff88.sol

```
3    pragma solidity ^0.8.27;
```

Recommendation

Pin the compiler (`pragma solidity 0.8.27;`) in future deployments so the reviewed and deployed bytecode are always produced by the same compiler.

I-02 — Entire Supply Minted to a Single Externally Owned Account

INFORMATIONAL **Status: ACKNOWLEDGED / OPEN (design decision)**

Description

The constructor mints all 30,000,000,000 LWT to the `recipient` argument, which at deployment was the deployer's own wallet `0x24AFd1BAbAf101A80a6000c2c865aE6261BA8183` (an EOA — no code at that address). On-chain reads at block 130,804,755 show that this wallet still holds 100.00% of the supply and that the token has exactly one holder and one transfer (the mint). There is no vesting, lock or multi-signature control over this balance. While this is not a code vulnerability, it means the token's circulating supply, liquidity provision and price are entirely under the control of one private key, and loss or compromise of that key would affect the whole supply.

Relevant code — contract-f61dacff88.sol

```
13        _mint(recipient, 30000000000 * 10 ** decimals());
```

Recommendation

Move the treasury balance to a multisignature wallet (for example 3-of-5) and publish the distribution plan; place team and reserve allocations in time-locked vesting contracts so holders can verify the schedule on chain. Consider using a hardware-backed key for the current holder until the transfer is made.

I-03 — `permit` Front-Running Can Make a Bundled Transaction Revert

INFORMATIONAL **Status: NO ACTION NEEDED (known EIP-2612 property)**

Description

A permit signature is valid for anyone to submit. If an integrator contract calls `permit` and then `transferFrom` in one transaction, a third party can copy the signature from the mempool and submit it first; the integrator's own `permit` call then reverts with `ERC2612InvalidSigner` because the nonce has advanced, and the whole bundled transaction fails even though the allowance was set exactly as intended. Test T31 reproduced this: the allowance is

correct after the front-run and the second submission reverts; no tokens can be redirected. This is a documented property of EIP-2612 and affects only the integrator's user experience.

Relevant code — @openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol

```
51      if (block.timestamp > deadline) {
52          revert ERC2612ExpiredSignature(deadline);
53      }
54
55      bytes32 structHash = keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value,
_useNonce(owner), deadline));
56
57      bytes32 hash = _hashTypedDataV4(structHash);
58
59      address signer = ECDSA.recover(hash, v, r, s);
60      if (signer != owner) {
61          revert ERC2612InvalidSigner(signer, owner);
62      }
```

Recommendation

Integrators should wrap the `permit` call in `try/catch` (or check `allowance` first) and continue when the allowance is already sufficient, as recommended by OpenZeppelin.

I-04 — No NatSpec Documentation and No License Shown on BscScan

INFORMATIONAL **Status: ACKNOWLEDGED / OPEN**

Description

The application contract carries an MIT SPDX identifier in the source but the BscScan verification lists the license as "-NA-", and the contract has no NatSpec (`@title`, `@notice`, `@param`) describing the token's purpose, the meaning of the supply figure or the intended recipient. This has no security impact but reduces transparency for holders and integrators reading the explorer.

Recommendation

Add NatSpec to the contract and select the matching license when verifying future deployments; optionally update the BscScan token information page (project links, logo, description).

I-05 — Supply Literal Written Without Digit Separators

INFORMATIONAL **Status: ACKNOWLEDGED / OPEN**

Description

The supply is written as `30000000000`, which Slither flags (too-many-digits) because an extra or missing zero is easy to miss in review. We verified on chain that `totalSupply()` equals exactly 30,000,000,000 × 10^{18}, matching the BscScan "Max Total Supply" of 30,000,000,000 LWT, so the deployed value is correct.

Relevant code — contract-f61dacff88.sol

```
13      _mint(recipient, 30000000000 * 10 ** decimals());
```

Recommendation

Write `30_000_000_000 * 10 ** decimals()` or `30e9 * 1e18`, or declare a named constant `INITIAL_SUPPLY`, in future versions.

Gas Optimizations

G-01 — Contract Deployed With the Optimizer Disabled

GAS **Status: ACKNOWLEDGED / OPEN (immutable contract)**

Description

The verified settings show the optimizer switched off. Compiling the same source with the optimizer enabled (200 runs) produces a 3,566-byte runtime instead of 6,802 bytes and reduces deployment gas from 1,662,944 to 942,193 (−43%). Runtime savings are small because the hot paths are already simple: in our measurements `transfer` to a new holder costs 52,143 gas unoptimized against 51,579 optimized, and `approve` 46,916 against 46,354 (about 1% per call).

Operation	Deployed (optimizer off)	Optimizer on, 200 runs	Saving
Deployment	1,662,944 gas	942,193 gas	720,751 (43.3%)
Runtime bytecode size	6,802 bytes	3,566 bytes	3,236 bytes

`transfer` (new holder)	52,143 gas	51,579 gas	564 (1.1%)
`transfer` (existing holder)	35,043 gas	34,479 gas	564 (1.6%)
`approve`	46,916 gas	46,354 gas	562 (1.2%)
`permit`	76,941 gas	—	—

Recommendation

Enable the optimizer for future deployments. No action is possible on the deployed contract and the per-transaction cost to users is negligible.

Centralization & Privileged Roles

The contract defines no roles. There is no owner, no minter, no pauser, no fee setter, no blacklist and no upgrade path, so no address can change the token's behaviour after deployment. The only trust point is off-chain: who controls the wallet that received the supply.

Role	Powers	Safeguards / Risk
Contract owner / admin	None — no such role exists in the code	Rules are fixed for ever; cannot be abused or rescued.
Supply recipient (`0x24AF…8183`, deployer EOA)	Holds 100% of the 30 B supply; can sell, distribute, add or remove liquidity at will	No lock, vesting or multisig observed on chain (I-02). Key compromise = loss of entire supply.
Any holder	Transfer, approve, sign permits for own balance only	Standard ERC-20 guarantees.

> **Engineer Note:** EtherAuthority recommends that the treasury balance be moved to a multisignature wallet (for example 3-of-5), that team and reserve allocations be placed in publicly verifiable vesting or locker contracts, and that any liquidity-pool tokens be locked or burned, so holders can verify the distribution independently of the team.

Deployment Verification

Because the audit was performed on an already-deployed contract, we verified that the reviewed source is what is running on chain rather than relying on the explorer's label alone.

Check	Result
Source verification on BscScan	"Source Code Verified — Exact Match", compiler v0.8.27+commit.40a35a09, optimizer off, Standard-JSON input with 21 files.
Independent recompilation	Reviewed source + OpenZeppelin 5.7.0 from npm, solc 0.8.27, optimizer off → 6,802-byte runtime, identical length to `eth_getCode` (6,802 bytes).
Byte comparison	167 differing bytes out of 6,802, all inside the five EIP-712 immutable slots (cached contract address, chain id 56, cached domain separator, hashed name, hashed version — values the compiler fills at deployment) and the 51-byte CBOR metadata hash (which encodes the original file name). No opcode differs.
Constructor argument	`recipient = 0x24AFd1BAbAf101A80a6000c2c865aE6261BA8183` (decoded from creation code; same as the deployer).
Creation transaction	`0xc32b79e8bc9fd2b253a20ee4ab6e7b6d8d04af94963087812c8b4d46937b5768`, block 130,539,122, Oct 29, 2026 20:43:18 UTC, status Success, one `Transfer(0x0 → recipient, 30,000,000,000 LWT)`.
On-chain state (block 130,804,755)	`totalSupply` = 30,000,000,000 × 10^{18}; `decimals` = 18; `balanceOf(recipient)` = 100.00% of supply; `nonces(recipient)` = 0; holders = 1; transfers = 1; contract BNB balance = 0.
Deployer account type	Externally owned account (no code).

Executed Test Summary

Tests	Coverage	Result
T01–T06	Deployment: metadata, supply, single mint event, no privileged functions in ABI, zero-address recipient rejected	6 / 6 pass
T07–T13	Transfers: exact amounts, over-balance, zero address, zero value, self-transfer, transfer to contract, supply invariant	7 / 7 pass
T14–T20	Approvals and `transferFrom`: allowance accounting, infinite allowance, zero spender, return values, approval race documented	7 / 7 pass
T21–T32	Permit: domain, relayed permit, nonce, replay, expiry, tampering, wrong chain id, high-s malleability, garbage signature, front-run, spend	12 / 12 pass
T33	300 random transfers among five holders: supply conserved, over-balance transfers revert	1 / 1 pass

Final Verdict

Status: PASSED — NO SECURITY ISSUES REQUIRING REMEDIATION. The LF Wallet Token contract is a minimal, unmodified composition of audited OpenZeppelin 5.7.0 components with a fixed supply and no privileged roles. No Critical, High or Medium vulnerability was found by manual review, static analysis or 33 executed tests, and the deployed bytecode was confirmed to be produced by the reviewed source. The two Low findings are inherent ERC-20 properties and the Informational items are design and documentation notes; none allows loss or theft of holder funds through the contract. Holders should note that the full supply is controlled by one wallet (I-02), which is a trust consideration outside the contract code.

Disclaimer

This report is a Final Audit Report for the contract deployed at the address identified in the Audit Scope. It reflects the state of the verified source code and on-chain state at the time of review (October 31, 2026). It is not an endorsement of the project, its team, its tokens or its business model, and it is not investment, legal or financial advice.

A smart-contract audit reduces risk but cannot guarantee the absence of vulnerabilities. The review was time-boxed and limited to the files listed in scope; the OpenZeppelin library files were checked for integrity against the official release and reviewed only along the code paths this token uses. Front-end, off-chain services, key management, token distribution, liquidity arrangements and future code deployed at other addresses were not reviewed. Test results describe behaviour observed in the executed environment (a local EVM) and on-chain reads from a public BNB Smart Chain node and may not cover every on-chain condition. EtherAuthority accepts no liability for losses arising from the use of the audited contract. Users and investors should perform their own due diligence.



Smart Contract Security & Blockchain Development

www.etherauthority.io