SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Table of Contents

ShopinX Token - Smart Contract Security Audit Report

Field	Value
Report Title	Smart Contract Security Audit - ShopinX Token
Audit Type	Smart Contract
Version	v1.3
Date	2026-06-15
Classification	Public
Prepared by	SolidProof.io
Prepared for	ShopinX

Document Control

Revision History

Version	Date	Author	Changes	Reviewed By
v1.0	2026-06-15	SolidProof.io	Initial release (re-audit of ShopinXToken v3)	Product owner

Confidentiality Notice

This document is classified as **Public** and is intended primarily for the use of ShopinX and SolidProof.io. Redistribution should preserve the document in its entirety and attribute it to SolidProof.io.

Disclaimer

This report is provided on an "as-is" basis and reflects the state of the smart contracts at the time of the audit. It does not constitute legal, financial, or investment advice.

Limitations:

- Security audits are time-boxed engagements and cannot guarantee the absence of all vulnerabilities.
- This audit does not replace other security measures such as bug bounty programs, on-chain monitoring, incident response planning, or ongoing security reviews.
- Findings are based on the code, configurations, and documentation provided by the client at the time of engagement. Changes made after the audit may introduce new risks.
- The assessment does not cover economic, game-theoretic, or market manipulation attack vectors unless explicitly stated in the scope.
- Third-party dependencies and integrations are assumed to function as documented unless explicitly included in the audit scope.
- This report is owned by SolidProof.io and prepared for ShopinX. Distribution to third parties should preserve attribution.

Liability:

SolidProof.io assumes no liability for any losses, damages, or exploits arising from the use of the audited smart contracts, whether or not the vulnerabilities were identified in this report.

Executive Summary

Engagement Overview

SolidProof.io was engaged by ShopinX to perform a smart contract security audit of the ShopinX Token (`src/v3/ShopinXToken.sol`). This report is a re-audit of the v3 contract, re-assessing all findings raised during the previous reviews and validating the remediation work delivered in this version. The structured, per-item findings database is maintained separately in `audit-findings.json`; this document is the human-readable deliverable.

The ShopinX Token is an ERC20 token with a fixed, non-mintable supply minted once at deployment, burnable balances, EIP-2612 permit approvals, owner-controlled time locks, and a built-in cliff-plus-step vesting mechanism. It is built on top of well-established OpenZeppelin components, and the project-specific logic is concentrated in the transfer gate that enforces the locks and vesting schedules.

Overall Security Rating

Rating	Description
Critical	Immediate action required - contracts are at risk of exploitation
Needs Improvement	Significant issues identified - remediation required before deployment
Moderate	Some issues found - manageable with timely fixes
Secure	No critical or high issues - minor improvements recommended
Excellent	Robust security posture - best practices observed

This engagement's rating: Secure

v3 is a genuine remediation pass. Both previously reported high-severity issues are fixed, all four medium issues are addressed (two by code changes, two by an explicit design decision plus documentation), and the low and optimization items are resolved. The remaining open items are one low-severity correctness note and informational product or operational decisions. None of the remaining items prevent a safe deployment, provided the operational recommendations are followed.

Findings Summary

Severity	Count	Open	In Progress	Verified	Accepted Risk
Critical	0	0	0	0	0
High	2	0	0	2	0
Medium	4	0	0	4	0
Low	4	1	0	3	0
Optimization	4	0	0	4	0
Informational	10	5	1	2	2
Total	24	6	1	15	2

Note on statuses: "Verified" indicates the auditor independently validated the fix. "Accepted Risk" covers items that are expected standard behaviour or accepted disclosures requiring no code change. "In Progress" covers items partially addressed in code with a remaining external action.

Key Observations

- Both previously reported high-severity issues are fixed. Freezes are now time-bounded (capped at a maximum lock duration of four years) and reversible via `removeLock`, and transfer locks now reserve only the transferred tranche rather than the recipient's entire balance.
- All four medium-severity issues are addressed. Two were fixed in code (view consistent with the transfer gate, and `Ownable2Step` two-step ownership), and two were resolved through explicit, documented design decisions (the fungible vesting reserve and the burn-while-locked semantics).
- The remaining open items are one low-severity, owner-operated correctness note (the amount-lock overwrite) and informational product or operational decisions. Before deployment, we recommend placing the owner behind a multisig and timelock, publishing the administrative addresses, verifying the EIP-712 domain separator post-deployment, and confirming the intended vesting and burn semantics.

Scope of Work

Target Description

The ShopinX Token implements an ERC20 token with a fixed, non-mintable supply minted once at deployment. It supports burnable balances, EIP-2612 permit approvals for signature-based allowances, owner-controlled time locks, and a built-in cliff-plus-step vesting mechanism. The token is built on OpenZeppelin's ERC20, ERC20Burnable, ERC20Permit, and Ownable2Step components. The project-specific logic is concentrated in the transfer gate (`_update`) that enforces both the address freeze / amount lock and the vesting schedules on every non-mint transition.

Reproducibility Metadata

Field	Value
Build/Test Environment	Foundry toolchain, Solidity compiler pinned to `0.8.27`
Tooling Version Lock	OpenZeppelin Contracts 5.6.1; Slither 0.11.5; Aderyn 0.6.8
Test Command Set	`forge build`, `forge test`, `slither .`, `aderyn .`
Analysis Time Window	April 2026 – Juni 2026
Constraints and Assumptions	Static analysis was run on the clean source; the inlined dependencies in the flattened file were excluded so results reflect only the contract under review.

Contracts in Scope

#	Contract / File	LoC	Description
1	`src/v3/ShopinXToken.sol`	~215	ERC20 token with owner-controlled locks and cliff-plus-step vesting

In Scope

- `src/v3/ShopinXToken.sol`, including the transfer gate, the full-address freeze / amount lock functionality (`setLock`, `removeLock`, `transferWithLock`, `isLocked`), the vesting functionality (`transferWithVesting`, `vestedAmount`), and the availability view (`availableToTransfer`).

Out of Scope

- Frontend and backend application code.
- Third-party contracts and libraries (OpenZeppelin Contracts 5.6.1 is assumed correct as a widely reviewed, version-pinned dependency).
- Economic and tokenomics modeling and game-theoretic analysis.
- Deployment scripts and migration tooling.

- Any contracts other than `src/v3/ShopinXToken.sol`.

Methodology & Tools

Approach Overview

The audit followed a multi-layered approach combining automated static analysis, manual expert review, and adversarial reasoning specific to smart contract security, with a dedicated remediation-verification pass against the previously reported findings.

Techniques Applied

Technique	Applied
Manual Code Review (line-by-line)	[x]
Static Analysis (SAST)	[x]
Formal Verification	[ ]
Symbolic Execution	[ ]
Fuzzing / Property-Based Testing	[ ]
Dependency & Supply Chain Analysis	[x]
Threat Modeling	[x]
Gas / Performance Optimization Review	[x]

Tools Used

Tool	Purpose	Version
Slither	Static analysis (Solidity)	0.11.5
Aderyn	Static analysis (Solidity)	0.6.8
Foundry	Testing & build framework	[version to be provided]

Review Process

1. **Kick-off and Documentation Review** - Establish architecture understanding, trust assumptions, and threat model scope.
2. **Automated Analysis Pass** - Execute the toolchain (Slither, Aderyn) and preserve raw outputs for traceability.
3. **Manual Expert Review** - Perform line-by-line review and adversarial reasoning against the lock and vesting flows.
4. **Exploitability Validation** - Validate practical exploit paths and business impact where feasible.
5. **Client Triage Session** - Confirm context, intended behavior, and remediation feasibility.
6. **Draft Reporting** - Document findings with technical evidence and client-facing risk statements.
7. **Remediation Verification** - Validate fixes against the original root cause and residual risk.
8. **Final Delivery** - Provide the report with status lifecycle and release-readiness statement.

Severity & Status Definitions

House Standard Risk Model

All findings use a unified house standard with mandatory mappings:

- **House Severity**: Critical / High / Medium / Low / Informational (Optimization findings are gas and code-quality recommendations with no direct security impact and are tracked separately in this report).
- **CVSS v3.1 Base Score**: Recorded for all security-impacting findings.
- **CWE Mapping**: Recorded for all findings where a matching CWE exists.

House Severity	CVSS Base Range	Typical Business Impact
Critical	9.0 - 10.0	Immediate risk to funds, governance, or protocol integrity
High	7.0 - 8.9	Material risk requiring remediation before release
Medium	4.0 - 6.9	Contained but meaningful risk; remediation required on schedule
Low	0.1 - 3.9	Limited exploitability or low-impact weakness
Informational	0.0	Quality or hardening recommendation without direct security impact

Status Lifecycle

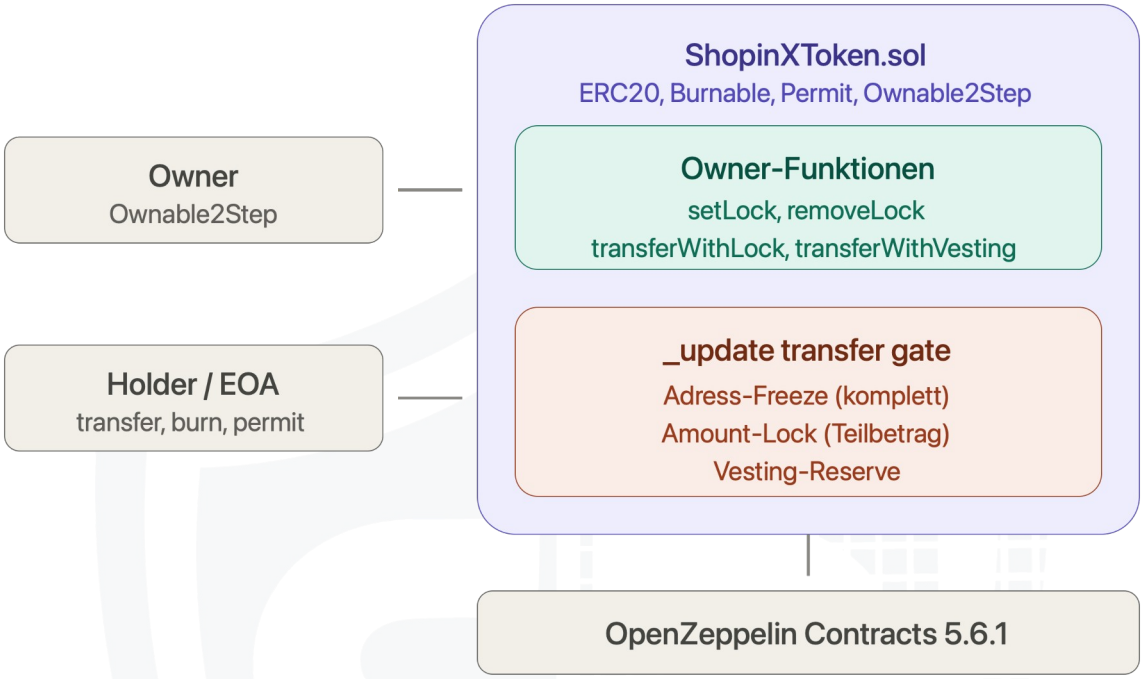
Status	Definition	Required Evidence
Open	Finding confirmed and not remediated.	Reproduction steps and affected location
In Progress	Fix work has started but is not yet complete.	Linked work item / partial change
Resolved	Client submitted a fix and claims closure.	Fix commit / PR reference
Verified	Auditor independently validated the fix.	Verification method and result
Accepted Risk	Client accepts residual risk with justification.	Formal risk acceptance statement

Logical Understanding / Contract Architecture

System Overview

The ShopinX Token is a single-contract system. It inherits from OpenZeppelin's ERC20, ERC20Burnable, ERC20Permit, and Ownable2Step. The full supply is minted once at deployment to a recipient address and cannot be increased afterwards. The owner holds administrative powers over locks and vesting, but cannot mint new tokens and cannot move tokens already held by other addresses.

Architecture Diagram



Contract Interactions & Data Flow

The external entry points are the standard ERC20 interface (`transfer`, `transferFrom`, `approve`), the ERC20Burnable interface (`burn`, `burnFrom`), the ERC20Permit interface (`permit`), the Ownable2Step interface, and the project-specific owner functions (`setLock`, `removeLock`, `transferWithLock`, `transferWithVesting`) plus the views (`isLocked`, `availableToTransfer`, `vestedAmount`). Every non-mint transition routes through `_update`, which enforces the full-address freeze first and then reserves the amount lock plus the still-unvested vesting amount against the sender's balance. The public `availableToTransfer` view applies the same freeze-then-reserve logic so it matches what a transfer will actually permit.

Contract Analysis (Smart Contract Analysis Statement)

The ShopinX Token contract implements an ERC20 token with a fixed, non-mintable supply minted once at deployment, burnable balances, EIP-2612 permit approvals for signature-based allowances, owner-controlled time locks, and a built-in cliff-plus-step vesting mechanism. The token is built on top of well-established OpenZeppelin components, and the project-specific logic is concentrated in the transfer gate that enforces the locks and the vesting schedules.

Ownership Privileges

Ownership of the contract is retained under a two-step ownership-transfer process, with the ability to renounce ownership deliberately disabled so the contract cannot be left without an administrator. We recommend that ownership be assigned to a multi-signature wallet, ideally fronted by a timelock, given the operational powers described below. The owner retains the following privileges:

- Freezing the entire balance of any address for a bounded period through the full-address lock.
- Distributing tokens together with an amount-based lock that applies only to the transferred tranche.
- Creating a cliff-plus-step vesting schedule for any address.
- Removing a previously applied full-address freeze to correct mistakes.

The owner is constrained as follows:

- The owner cannot create new tokens after deployment, as the total supply is fixed at the initial mint and there is no minting function.
- The owner cannot move or seize tokens already held by other addresses, since administrative transfers are funded only from the owner's own balance.
- The owner cannot lock an address indefinitely, as the maximum lock duration is capped at four years.
- The owner cannot alter or cancel a vesting schedule once it has been created.

Security Features

- The contract builds on widely used, community-reviewed OpenZeppelin token, ownership, and signature components rather than reimplementing this functionality from scratch.
- The compiler version is pinned to an exact release, and the Solidity 0.8 series provides built-in protection against arithmetic overflow and underflow.
- Ownership uses a two-step handover and the renounce path is disabled, which reduces the risk of accidental or irreversible loss of administrative control.
- Administrative locks are bounded in time and reversible, every privileged function validates its inputs, and the public availability view is kept consistent with the rules enforced on actual transfers.

Trust Boundaries

Boundary	Trust Level	Notes
User / EOA -> Contract	Untrusted	All input validated at the transfer gate
Contract internal (`_update`)	Trusted	Enforces freeze, amount lock, and vesting reserve
Admin / Owner -> Contract	Privileged	Access-controlled lock and vesting functions; broad but bounded

External Dependencies & Integrations

Dependency	Type	Risk Level	Notes
OpenZeppelin Contracts	Library	Low	Version-pinned to 5.6.1; widely reviewed

Codebase Analysis Summary & Metrics

Code Overview

Metric	Value
Total Lines of Code (LoC)	~215
Language(s)	Solidity `0.8.27`
Number of Contracts	1 (`ShopinXToken`)
Number of External Functions	6 project-specific (`setLock`, `removeLock`, `transferWithLock`, `isLocked`, `transferWithVesting`, `availableToTransfer`) plus inherited ERC20 / Burnable / Permit / Ownable2Step
Test Coverage (%)	/
Number of External Imports	4 (ERC20, ERC20Burnable, ERC20Permit, Ownable2Step)

Code Quality Assessment

Category	Rating	Notes
Code Readability	4 / 5 - Strong	Clear structure; intent documented in-code after v3
Documentation / NatSpec	4 / 5 - Strong	Sentinels and design decisions now documented in the source
Error Handling	4 / 5 - Strong	Input validation added across privileged functions
Naming Conventions	4 / 5 - Strong	Named constants introduced for supply and bounds
Modularity / Separation	4 / 5 - Strong	Logic concentrated in a single, well-scoped transfer gate
Gas Efficiency	4 / 5 - Strong	External visibility, named constants, packed vesting struct
Input Validation	4 / 5 - Strong	Zero-address, positive-amount, and upper-bound checks added
Access Control Patterns	4 / 5 - Strong	Ownable2Step; renounce disabled; multisig + timelock recommended

Rating Scale: 5 - Excellent | 4 - Strong | 3 - Adequate | 2 - Needs Improvement | 1 - Insufficient

Dependency Health

Dependency	Version	Pinned?	Notes
OpenZeppelin Contracts	5.6.1	Yes	ERC20, ERC20Burnable, ERC20Permit, Ownable2Step

Static Analysis Results

Both analyzers were run on the clean source (the inlined dependencies in the flattened file were excluded so the results reflect only the contract under review).

Slither (0.11.5) produced ten results, none of which are actionable security defects:

- Reliance on `block.timestamp` - accepted disclosure; the logic runs at day-level granularity where validator influence is negligible.
- Weak-PRNG on the within-step modulo - confirmed false positive; the modulo is a deterministic time interpolation, now documented in the code.
- Strict equality on a zero `totalAmount` - confirmed false positive; zero is the idiomatic sentinel for an unset schedule, now documented in the code.
- Divide-before-multiply on `completedSteps` - by design; that division is the intended floor count of completed steps. The precision-sensitive interpolation was reordered to multiply before dividing.

No reentrancy, arbitrary-send, uninitialized-state, or access-control detector fired.

Aderyn (0.6.8) produced one low-severity result: the centralization notice on the owner-only functions, which is inherent to the trusted-owner design and disclosed accordingly. The unspecific pragma, PUSH0, large-literal, literal-instead-of-constant, and public-not-external findings from the previous version are all cleared.

Findings Overview

Summary Table

ID	Title	Severity	Status	Location
H-01	Owner can irreversibly freeze any holder via `setLock`	High	Verified	`ShopinXToken.sol:64-76`
H-02	`transferWithLock` freezes the recipient's entire balance	High	Verified	`ShopinXToken.sol:79-92`
M-01	`availableToTransfer` view inconsistent with `_update`	Medium	Verified	`ShopinXToken.sol:153-173,177-215`
M-02	Single-step Ownable with broad operational powers	Medium	Verified	`ShopinXToken.sol:7,9`
M-03	Vesting lien enforced as a fungible reserve	Medium	Verified	`ShopinXToken.sol:21,199-212`
M-04	Locked and undervested holders cannot burn their tokens	Medium	Verified	`ShopinXToken.sol:183-212`
L-01	Floating Solidity pragma and default PUSH0 target	Low	Verified	`ShopinXToken.sol:2`
L-02	Missing input validation on privileged functions	Low	Verified	`ShopinXToken.sol:64,79,101`

ID	Title	Severity	Status	Location
L-03	`renounceOwnership` exposed and would be destructive	Low	Verified	`ShopinXToken.sol:57-59`
L-04	`transferWithLock` overwrites an existing amount-lock	Low	Open	`ShopinXToken.sol:73-76,88-89`
O-01	Mark public administrative and view functions as external	Optimization	Verified	`ShopinXToken.sol:64,73,79,94,101,153`
O-02	Replace repeated numeric literals with named constants	Optimization	Verified	`ShopinXToken.sol:11-16`
O-03	Prefer multiplication before division to reduce precision loss	Optimization	Verified	`ShopinXToken.sol:140-147`
O-04	Pack `VestingInfo` fields to reduce storage writes	Optimization	Verified	`ShopinXToken.sol:26-32`
I-01	Vesting schedules are write-once and cannot be amended	Informational	Open	`ShopinXToken.sol:114`
I-02	Vesting releases tokens continuously within each step	Informational	Open	`ShopinXToken.sol:144-149`
I-03	`permit` allowances can still be signed while an account is locked	Informational	Accepted Risk	`ShopinXToken.sol:177-215`
I-04	Reliance on `block.timestamp` for time comparisons	Informational	Accepted Risk	`ShopinXToken.sol:66,86,132,155,188`
I-05	Modulo used for interpolation flagged as weak randomness	Informational	Verified	`ShopinXToken.sol:146`
I-06	Strict equality on `totalAmount == 0` flagged as dangerous	Informational	Verified	`ShopinXToken.sol:114,130`
I-07	Verify EIP-712 domain separator after deployment	Informational	Open	`ShopinXToken.sol:49,51`
I-08	Recipient and initial owner addresses should be documented	Informational	In Progress	`ShopinXToken.sol:46-47`
I-09	Initial supply changed from the originally audited figure	Informational	Open	`ShopinXToken.sol:12,53`
I-10	Unchecked downcasts when writing the vesting struct	Informational	Open	`ShopinXToken.sol:116-121`

Detailed Findings

F-H-01: Owner can irreversibly freeze any holder via `setLock`

Field	Value
ID	H-01
Severity	High
Status	Verified
Location	`src/v3/ShopinXToken.sol:64-76`
Category	Access Control / Centralisation
CVSS v3.1	7.5
CWE ID	CWE-284 (Improper Access Control)
Exploit Preconditions	Attacker holds the owner key
Business Impact	Permanent, unrecoverable freeze of any holder's tokens amounts to an irreversible on-chain blacklist
Remediation Reference	v3 remediation - `MAX_LOCK_DURATION` cap and `removeLock`

Description

`setLock` accepted any `unlockTime` strictly greater than `block.timestamp`, with no upper bound. Combined with the invariant that a lock could only ever be extended, never reduced, the owner could call `setLock(victim, type(uint256).max)` and freeze every token the victim currently held as well as every token they ever received afterwards, because `_update` uses `lockUntil[from]` as the authoritative gate on every outgoing transfer. The freeze could not be reduced or removed by any party afterwards, not even the owner, because of the monotonic-only rule.

Impact

An accidental or malicious call permanently disables all outgoing transfers for the affected address, with no recovery path. In practice this is an irreversible blacklist over arbitrary holders.

Exploit Scenario

The owner (or a compromised owner key) calls `setLock(victim, type(uint256).max)`. Every subsequent outgoing transfer from `victim` reverts in `_update`, indefinitely, and no on-chain action can undo it.

Recommendation

Introduce an upper bound on `unlockTime` (for example `block.timestamp + MAX_LOCK_DURATION` capped at a few years) and add an owner-controlled `reduceLock` / `removeLock` function, itself guarded by a multisig and a timelock. Publicly disclose that `setLock` is a trusted administrative primitive, and ensure marketing does not claim the token cannot be blacklisted.

Remediation & Verification

Resolved and verified. `setLock` now caps the freeze at `MAX_LOCK_DURATION` of four years and rejects the zero address (lines 64-70), and a new `removeLock` lets the owner clear a freeze (lines 73-76). The freeze is therefore bounded and reversible rather than permanent. `setLock` remains a trusted administrative primitive and should still be disclosed as such.

F-H-02: `transferWithLock` freezes the recipient's entire pre-existing and future balance

Field	Value
ID	H-02
Severity	High
Status	Verified
Location	`src/v3/ShopinXToken.sol:79-92`
Category	Logic Error / Access Control
CVSS v3.1	7.5
CWE ID	CWE-284 (Improper Access Control)
Exploit Preconditions	Owner sends any amount with a far-future unlock time
Business Impact	A single wei sent with a lock freezes an exchange or market-maker wallet entirely
Remediation Reference	v3 remediation - amount-based lock

Description

`transferWithLock` stored the lock as a timestamp on the recipient's address (`lockUntil[to] = unlockTime`). Because `_update` checked that timestamp for every outgoing transfer, the lock extended to the recipient's whole balance at that moment and to any tokens they later received from third parties, not just the amount transferred in the call. A single wei of SPX sent to a market-maker, exchange hot wallet, or retail holder with a far-future `unlockTime` froze that address entirely for the duration of the lock.

Impact

Unintended, wide-blast-radius freezing of third-party balances, including balances unrelated to the amount actually distributed.

Exploit Scenario

The owner sends a small `transferWithLock` to an exchange hot wallet with a far-future unlock time. The exchange's entire SPX balance, plus all later deposits, becomes non-transferable until the lock expires.

Recommendation

Track locks as amounts rather than a single per-address timestamp. Increment `lockedAmount[account]` inside `transferWithLock` and enforce `balanceOf(from) - lockedAmount[from] >= value` in `_update`, or apply the lock only to the tranche transferred in the same call.

Remediation & Verification

Resolved and verified. `transferWithLock` now records an amount-based lock (`lockedAmount` and `lockedUntil`) and the gate only reserves that tranche, not the recipient's whole balance or later incoming tokens (lines 79-92, 194-212). See the separate low-severity note L-04 about the lock being overwritten rather than accumulated on repeated calls to the same address.

F-M-01: `availableToTransfer` view is inconsistent with the rules enforced in `_update`

Field	Value
ID	M-01
Severity	Medium
Status	Verified
Location	`src/v3/ShopinXToken.sol:153-173, 177-215`
Category	Logic Error
CVSS v3.1	5.3
CWE ID	CWE-684 (Incorrect Provision of Specified Functionality)
Exploit Preconditions	Address carries both a time-lock and a vesting schedule
Business Impact	UIs, routers, bots, and accounting tools mislead users about transferable balance
Remediation Reference	v3 remediation - unified freeze-then-reserve logic

Description

When an address carried both a time-lock (`lockUntil` in the future) and a vesting schedule, `availableToTransfer` evaluated only the vesting branch and could return a non-zero value, while `_update` still reverted the actual transfer because it checked `block.timestamp >= lockUntil[from]` first. Interfaces, routers, bots, and accounting tools relying on this view were misled about what could actually be moved. Additionally, `transferWithVesting` never inspected `lockUntil[to]` when writing a new schedule, which was a silent way to land an address in the inconsistent state.

Impact

Downstream tooling reports a transferable amount that transfers cannot honor, causing failed transactions and incorrect accounting.

Recommendation

Short-circuit `availableToTransfer` when the account is time-locked, regardless of whether a vesting schedule exists, so the view always reflects what `_update` will permit. Optionally surface whether the zero comes from a lock or a vesting lien.

Remediation & Verification

Resolved and verified. The view and the transfer gate now apply the same rules: both check the full-address freeze first and then subtract `amountLocked + vestingLocked` (`availableToTransfer` lines 153-173, `_update` lines 177-215), so the view matches what a transfer will actually permit.

F-M-02: Single-step Ownable with broad, irreversible operational powers

Field	Value
ID	M-02
Severity	Medium
Status	Verified
Location	`src/v3/ShopinXToken.sol:7, 9`
Category	Access Control
CVSS v3.1	6.5
CWE ID	CWE-284 (Improper Access Control)
Exploit Preconditions	Mistyped ownership transfer or single-key compromise
Business Impact	Loss or compromise of the single owner key has an outsized blast radius
Remediation Reference	v3 remediation - `Ownable2Step`

Description

The contract used a single-key, one-step `Ownable` for all administrative functions (lock creation, lock extension, vesting creation). A mistyped `transferOwnership` sent ownership to an unusable address in a single transaction, and there was no on-chain delay or co-signer requirement between a privileged call being initiated and it taking effect.

Impact

Given the impact of `setLock`, `transferWithLock`, and `transferWithVesting`, this access-control model was thinner than best practice for a production token, with no window for the community to react to unexpected administrative changes.

Recommendation

Replace `Ownable` with `Ownable2Step` so ownership transfer requires an explicit accept step. Assign ownership to a multisig (for example Safe) fronted by a `TimelockController` with a delay of at least 24 hours on the privileged functions.

Remediation & Verification

Resolved at the code level and verified. The contract now inherits `Ownable2Step` (lines 7, 9), so ownership transfer needs an explicit accept step. Placing the owner behind a multisig and a timelock remains an operational recommendation.

F-M-03: Vesting lien is enforced as a fungible reserve rather than a segregated escrow

Field	Value
ID	M-03
Severity	Medium
Status	Verified
Location	`src/v3/ShopinXToken.sol:21, 199-212`
Category	Design / Logic
CVSS v3.1	4.3
CWE ID	CWE-664 (Improper Control of a Resource Through its Lifetime)
Exploit Preconditions	Beneficiary holds unrelated balance alongside a grant
Business Impact	Off-chain tooling that expects segregated escrow may misaccount grants
Remediation Reference	v3 remediation - explicit in-code documentation

Description

The `_update` hook enforces `balanceOf(from) >= value + (v.totalAmount - vestedAmount(from))`. The result is a fungible reserve: any tokens in the account can satisfy the locked amount, and granted tokens are not distinguished from pre-existing or later-received tokens. This does not freeze unrelated balance, but it couples the accounting of the grant to the beneficiary's wider balance, and it makes burning, slashing, or clawing back granted tokens in isolation impossible.

Impact

Off-chain tooling that expects vesting grants to be held in a separate escrow may misinterpret the beneficiary's transferable balance.

Recommendation

Consider tracking the vesting lien on a dedicated counter (for example `lockedVestingBalance[account]`) decremented as vesting unlocks, or deploy a dedicated vesting escrow per beneficiary (such as OpenZeppelin's `VestingWallet`). At minimum, document the behaviour explicitly so investors and integrators understand it.

Remediation & Verification

Addressed by documentation and verified. The lien is still a fungible reserve against the whole balance, but this is now stated explicitly in the code (lines 21 and 199). A segregated per-beneficiary escrow was not adopted, which is acceptable given the explicit disclosure. Mirror the same note in the external documentation so integrators are aware.

F-M-04: Locked and undervested holders cannot burn their tokens

Field	Value
ID	M-04
Severity	Medium
Status	Verified
Location	`src/v3/ShopinXToken.sol:183-212`
Category	Logic Error / Design
CVSS v3.1	4.0
CWE ID	CWE-684 (Incorrect Provision of Specified Functionality)
Exploit Preconditions	Holder is under an active lock or has an undervested grant
Business Impact	Advertised "burnable" behaviour is conditional in an undocumented way
Remediation Reference	v3 remediation - burns skip the freeze; documented

Description

`_update` applied the lock and vesting checks to every non-mint transition, including `_burn` (which routes through `_update` with `to == address(0)`). A holder under an active lock or with an undervested grant therefore could not exercise the `burn` / `burnFrom` interface inherited from `ERC20Burnable`, contradicting the common expectation that holders can always destroy tokens they own.

Impact

The advertised "burnable" behaviour was conditional in a way that was not documented, which could confuse users and integrators.

Recommendation

Decide explicitly whether burning should remain possible while a holder is locked or undervested. If yes, special-case `to == address(0)` inside `_update`. If burns should be prohibited during a lock, drop `ERC20Burnable` from the interface or document the restriction prominently.

Remediation & Verification

Addressed as a deliberate, documented design and verified. Burns now skip the full-address freeze, so a frozen holder can still destroy tokens (lines 183-191). Amount-locked and still-unvested tokens remain non-burnable because the reserve check also applies to burns (lines 206-212); this is the intended behaviour that only unlocked tokens can be burned, and it is documented in the code. Confirm this matches the product intent and restate it in the user-facing documentation.

F-L-01: Floating Solidity pragma and default PUSH0 target

Field	Value
ID	L-01
Severity	Low
Status	Verified
Location	`src/v3/ShopinXToken.sol:2`
Category	Configuration / Coding Standard
CVSS v3.1	2.0
CWE ID	CWE-710 (Improper Adherence to Coding Standards); SWC-103
Exploit Preconditions	Deployment on a chain that does not support PUSH0
Business Impact	Non-reproducible builds and possible silent deployment failure
Remediation Reference	v3 remediation - pinned pragma

Description

The contract used a floating pragma `^0.8.27`, allowing compilation with any compatible compiler. Combined with the default modern solc target, the emitted bytecode uses `PUSH0`, which is not universally supported across EVM-compatible chains. Two rebuilds of the same source could produce different artifacts, and deployment could silently fail on chains that do not implement `PUSH0`.

Impact

Non-reproducible builds and potential silent deployment failures on non-PUSH0 chains.

Recommendation

Pin the compiler to an exact version (for example `pragma solidity 0.8.27;`). If deployment on a chain that does not support PUSH0 is planned, also set `evm_version` to `paris` (or earlier) in `foundry.toml`.

Remediation & Verification

Resolved and verified. The pragma is now pinned to an exact `0.8.27`, and Aderyn no longer reports the unspecific pragma or the PUSH0 opcode. One optional hardening remains: `foundry.toml` sets no `evm_version`, and `0.8.27` defaults to the Cancun target, so set `evm_version` explicitly if a chain that is not on Cancun is a deployment target.

F-L-02: Missing input validation on privileged functions

Field	Value
ID	L-02
Severity	Low
Status	Verified
Location	`src/v3/ShopinXToken.sol:64, 79, 101`
Category	Input Validation
CVSS v3.1	3.1
CWE ID	CWE-20 (Improper Input Validation)
Exploit Preconditions	Owner submits a malformed administrative call
Business Impact	Administrative mistakes can pollute storage or create unusable schedules
Remediation Reference	v3 remediation - added validation and bounds

Description

The privileged entry points did not validate operationally impactful inputs. `setLock` accepted `address(0)`; `transferWithLock` and `transferWithVesting` did not require `amount > 0`, so a zero-amount call silently consumed a vesting slot forever (given the write-once rule); and `transferWithVesting` lacked upper bounds on `cliffDays` and `stepDays`, allowing pathological schedule parameters.

Impact

Administrative errors could silently pollute storage, create permanently unusable vesting slots, or configure pathological schedules.

Recommendation

Add `require(to != address(0))`, `require(amount > 0)`, and sensible upper bounds on `cliffDays` and `stepDays` across the privileged functions.

Remediation & Verification

Resolved and verified. `setLock` now rejects the zero address and caps the lock at `MAX_LOCK_DURATION` (lines 65-67). `transferWithLock` now requires a non-zero recipient, a positive amount, and a bounded unlock time (lines 84-87). `transferWithVesting` now requires a non-zero recipient, a positive amount, and upper bounds on `stepDays` and `cliffDays` (lines 108-112). The bounded values also remove the earlier overflow-revert path inside `vestedAmount`.

F-L-03: `renounceOwnership` is exposed and would be destructive

Field	Value
ID	L-03
Severity	Low
Status	Verified
Location	`src/v3/ShopinXToken.sol:57-59`
Category	Access Control
CVSS v3.1	3.7
CWE ID	CWE-284 (Improper Access Control)
Exploit Preconditions	Accidental owner call to `renounceOwnership`
Business Impact	Permanent loss of all administrative paths, with locks and schedules frozen forever
Remediation Reference	v3 remediation - override to revert

Description

OpenZeppelin's `Ownable` exposes `renounceOwnership`, which sets the owner to the zero address. Given this contract's design, a renounced ownership would make every existing lock permanent and every existing vesting schedule frozen in place, with no path to correct a mistake.

Impact

An accidental renouncement would permanently disable every administrative path while leaving existing locks and vesting schedules immutable forever.

Recommendation

Override `renounceOwnership()` to always revert.

Remediation & Verification

Resolved and verified. `renounceOwnership` is overridden to revert (lines 57-59), so ownership can no longer be dropped by accident.

F-L-04: `transferWithLock` overwrites an existing amount-lock and there is no way to clear it

Field	Value
ID	L-04
Severity	Low
Status	Open
Location	`src/v3/ShopinXToken.sol:73-76, 88-89`
Category	Logic Error
CVSS v3.1	3.1
CWE ID	CWE-664 (Improper Control of a Resource Through its Lifetime)
Exploit Preconditions	Owner calls `transferWithLock` twice on the same address
Business Impact	An accidental second lock can release a previously locked tranche earlier than intended
Remediation Reference	Not yet remediated

Description

`transferWithLock` assigns `lockedAmount[to] = amount` and `lockedUntil[to] = unlockTime` rather than accumulating, so calling it twice on the same address replaces the earlier lock. Because the first tranche is already in the recipient balance, the previously locked tokens stop being reserved. `removeLock` only clears the full-address freeze, not the amount-based lock, so there is no direct way to undo an amount-lock except overwriting it with another transfer or waiting for expiry.

Impact

This is an owner-operated correctness and least-surprise issue rather than a third-party exploit, since the owner is trusted and can already manage freezes. The practical impact is that an accidental second lock can release a previously locked tranche earlier than intended.

Recommendation

Accumulate the locked amount (`lockedAmount[to] += amount`), store one record per tranche, or revert when an active lock already exists. Add a symmetric clear or reduce function for the amount-based lock that mirrors `removeLock`, and emit an event when a lock is cleared.

Remediation & Verification

Open. Raised against this version and not yet remediated.

F-O-01: Mark public administrative and view functions as external

Field	Value
ID	O-01
Severity	Optimization
Status	Verified
Location	`src/v3/ShopinXToken.sol:64, 73, 79, 94, 101, 153`
Category	Gas Optimization
CVSS v3.1	0.0
CWE ID	N/A
Remediation Reference	v3 remediation - external visibility

Description

Several administrative and view functions were declared `public` but were never called from within the contract. `public` adds a small gas overhead because arguments are copied from calldata into memory; `external` is the appropriate visibility when internal consumption is not needed.

Recommendation

Change the visibility of functions that are never called internally from `public` to `external`.

Remediation & Verification

Resolved and verified. The administrative and view entry points are now `external` (`setLock`, `removeLock`, `transferWithLock`, `isLocked`, `transferWithVesting`, `availableToTransfer`); `vestedAmount` stays `public` because it is called internally. Aderyn no longer reports the public-not-external pattern.

F-O-02: Replace repeated numeric literals with named constants

Field	Value
ID	O-02
Severity	Optimization
Status	Verified
Location	`src/v3/ShopinXToken.sol:11-16`
Category	Code Quality / Gas Optimization
CVSS v3.1	0.0
CWE ID	N/A
Remediation Reference	v3 remediation - named constants

Description

The literal `100` was repeated in percent validation and in the vesting maths, and the initial supply was written as an inline expression in the constructor. Promoting these to named constants improves readability, reduces bytecode duplication, and makes the values part of the contract's public API.

Recommendation

Declare `PERCENT_BASE = 100` and `INITIAL_SUPPLY = 1_000_000_000e18` and reuse them throughout.

Remediation & Verification

Resolved and verified. `PERCENT_BASE` and `INITIAL_SUPPLY`, along with the lock and schedule bounds, are now named constants (lines 11-16) and used throughout, and the supply uses scientific notation. Aderyn no longer reports the repeated literal or the large numeric literal.

F-O-03: Prefer multiplication before division to reduce precision loss

Field	Value
ID	O-03
Severity	Optimization
Status	Verified
Location	`src/v3/ShopinXToken.sol:140-147`
Category	Numeric Precision
CVSS v3.1	0.0
CWE ID	N/A
Remediation Reference	v3 remediation - reordered interpolation

Description

`vestedAmount` performed a division (`afterCliff / stepSeconds`) before multiplying by `stepPercent`, and computed `currentStepAmount` by dividing `totalAmount` by 100 before multiplying. Reordering to divide last keeps precision maximal and prevents rounding errors from accumulating across steps.

Recommendation

Reorder the vesting maths so all multiplications happen before any division for the interpolated component.

Remediation & Verification

Resolved for the precision-sensitive path and verified. The within-step interpolation now multiplies before dividing (line 147). Slither still reports a divide-before-multiply on `completedSteps` (lines 140 and 144), but that division is the intended floor count of completed steps and is correct by design.

F-O-04: Pack `VestingInfo` fields to reduce storage writes

Field	Value
ID	O-04
Severity	Optimization
Status	Verified
Location	`src/v3/ShopinXToken.sol:26-32`
Category	Gas Optimization / Storage Packing
CVSS v3.1	0.0
CWE ID	N/A
Remediation Reference	v3 remediation - packed struct

Description

`VestingInfo` declared five independent `uint256` fields. The stored values (token amounts bounded by total supply, day counts, and a percentage) all fit in smaller integer types, so the struct can be packed into fewer storage slots without loss of precision.

Recommendation

Use `uint128` for `totalAmount` and `uint32` for `startTime`, `cliffDays`, `stepDays`, and `stepPercent` so the struct fits in two storage slots instead of five.

Remediation & Verification

Resolved and verified. `VestingInfo` now uses `uint128` for the amount and `uint32` for the time and schedule fields (lines 26-32), packing into two slots. See the separate informational note I-10 on the unchecked downcasts used when writing the struct.

F-I-01: Vesting schedules are write-once and cannot be cancelled or amended

Field	Value
ID	I-01
Severity	Informational
Status	Open
Location	`src/v3/ShopinXToken.sol:114`
Category	Product / Operations Decision
CVSS v3.1	0.0
CWE ID	N/A
Remediation Reference	Product decision - unchanged

Description

The check `require(vesting[to].totalAmount == 0)` prevents any subsequent update to a vesting schedule once created, and the contract exposes no function to cancel or amend one. Whether this is acceptable depends on the product requirements around vesting.

Recommendation

If schedules must be revocable or amendable, add administrative `revokeVesting` / `amendVesting` functions guarded by a multisig and a timelock, and emit explicit events. If schedules are intended to be immutable, document that clearly.

Remediation & Verification

Open and unchanged. Schedules are still write-once (line 114) with no cancel or amend path. This remains a product decision.

F-I-02: Vesting releases tokens continuously within each step

Field	Value
ID	I-02
Severity	Informational
Status	Open
Location	`src/v3/ShopinXToken.sol:144-149`
Category	Product / Operations Decision
CVSS v3.1	0.0
CWE ID	N/A
Remediation Reference	Product decision - unchanged

Description

After the cliff ends, `vestedAmount` releases tokens continuously each second inside the current step. Classic cliff-plus-step schedules typically unlock tokens discretely at step boundaries. Whether the continuous behaviour is correct depends on what the project advertised.

Recommendation

Confirm which release pattern was advertised. For discrete step releases, drop the within-step term and only credit completed steps. For continuous linear release, a single duration parameter is simpler. Document the chosen pattern clearly on `vestedAmount` and in the README.

Remediation & Verification

Open and unchanged. The within-step term still releases tokens continuously each second (lines 144-149) rather than at discrete step boundaries. Confirm which pattern was advertised and document it.

F-I-03: `permit` allowances can still be signed while an account is locked

Field	Value
ID	I-03
Severity	Informational
Status	Accepted Risk
Location	`src/v3/ShopinXToken.sol:177-215`
Category	Standard Behaviour
CVSS v3.1	0.0
CWE ID	N/A
Remediation Reference	Expected ERC-2612 behaviour - no change required

Description

A locked holder can still produce valid EIP-2612 `permit` signatures because `permit` only updates the allowance. A subsequent `transferFrom` will revert in `_update` because of the lock. This matches the standard semantics of `permit`, which never promises that a downstream transfer will succeed.

Recommendation

No change is strictly required. If user experience is a concern for permit-based flows, consider overriding `permit` to revert when the signer is currently locked.

Remediation & Verification

Accepted risk. This is expected and unchanged. A frozen or locked holder can still sign a permit; the later `transferFrom` still reverts in the gate (lines 177-215). This matches ERC-2612 semantics, so no change is required unless the project wants to block signing during the frozen window for user-experience reasons.

F-I-04: Reliance on `block.timestamp` for time comparisons

Field	Value
ID	I-04
Severity	Informational
Status	Accepted Risk
Location	`src/v3/ShopinXToken.sol:66, 86, 95, 132, 155, 188`
Category	Time Dependence (disclosure)
CVSS v3.1	0.0
CWE ID	N/A; SWC-116
Remediation Reference	Accepted disclosure - no change required

Description

All lock checks, vesting progress checks, and availability checks use `block.timestamp`. Validators have a limited ability to influence this value, which is irrelevant at the day-level resolution used by the vesting and lock logic. Listed for completeness because it is an industry-standard disclosure.

Recommendation

No change required. Flag it in the project's security notes so investors and integrators are aware.

Remediation & Verification

Accepted risk and unchanged. The lock, freeze, and vesting checks still rely on `block.timestamp` (for example lines 66, 86, 132, 155, and 188), which Slither continues to report. At day-level granularity the validator influence is negligible.

F-I-05: Modulo used for interpolation flagged as weak randomness

Field	Value
ID	I-05
Severity	Informational
Status	Verified
Location	`src/v3/ShopinXToken.sol:146`
Category	Static Analysis False Positive
CVSS v3.1	0.0
CWE ID	N/A; SWC-120
Remediation Reference	v3 remediation - documented

Description

Static analysis reported `afterCliff % stepSeconds` as a weak pseudo-random number generator. The modulo here is purely a deterministic interpolation inside a vesting step, not a random value.

Recommendation

No action required beyond a short comment documenting the intent to prevent the finding from being reopened.

Remediation & Verification

Resolved by documentation and verified. A comment now states that the modulo is a deterministic interpolation and not randomness (lines 145-146). Slither still reports it, but it is a confirmed false positive.

F-I-06: Strict equality on `totalAmount == 0` flagged as dangerous

Field	Value
ID	I-06
Severity	Informational
Status	Verified
Location	`src/v3/ShopinXToken.sol:114, 130`
Category	Static Analysis False Positive
CVSS v3.1	0.0
CWE ID	N/A
Remediation Reference	v3 remediation - documented

Description

Static analysis flagged `v.totalAmount == 0` as a dangerous strict equality. Here zero is the default value for an uninitialised `VestingInfo`, so the comparison is the standard way to check whether a schedule has been set.

Recommendation

No action required beyond a short comment documenting the sentinel.

Remediation & Verification

Resolved by documentation and verified. A comment now documents that a zero `totalAmount` is the sentinel for an unset schedule (line 25). The strict-equality checks (lines 114 and 130) are the idiomatic sentinel test, so the Slither report is a confirmed false positive.

F-I-07: Verify EIP-712 domain separator after deployment

Field	Value
ID	I-07
Severity	Informational
Status	Open
Location	`src/v3/ShopinXToken.sol:49, 51`
Category	Post-Deployment Check
CVSS v3.1	0.0
CWE ID	N/A
Remediation Reference	Post-deployment action - pending

Description

`ERC20Permit` is initialised with the name "ShopinX Token", which matches the ERC20 name. A routine post-deployment check should confirm that the domain separator computed off-chain matches the one returned by the deployed contract, to rule out metadata or constructor-argument drift.

Recommendation

After deployment, compare the on-chain `DOMAIN_SEPARATOR` against the one produced by the reviewed source with the same constructor arguments. Investigate any mismatch before enabling `permit` for end users.

Remediation & Verification

Open and unchanged. The ERC20 name and the permit name still match (lines 49 and 51), so the post-deployment domain separator check still applies once the contract is live.

F-I-08: Recipient and initial owner addresses should be publicly documented

Field	Value
ID	I-08
Severity	Informational
Status	In Progress
Location	`src/v3/ShopinXToken.sol:46-47`
Category	Documentation / Transparency
CVSS v3.1	0.0
CWE ID	N/A
Remediation Reference	v3 remediation - in-code labels added

Description

The constructor receives the initial recipient of the full supply and the administrative owner as two arguments, with no on-chain labelling of what each address represents. Clear documentation of these addresses is essential for downstream trust evaluation given the broad administrative powers described elsewhere in this report.

Recommendation

In the project documentation, publish the exact addresses used for `recipient` (the initial holder of the full supply) and `initialOwner` (the administrative key), along with the nature of each address (EOA, multisig, custodian, treasury).

Remediation & Verification

Partially addressed. The constructor parameters now carry comments describing the recipient and the administrative owner (lines 46-47). Publishing the actual deployed addresses and their nature in the external documentation is still recommended.

F-I-09: Initial supply changed from the originally audited figure

Field	Value
ID	I-09
Severity	Informational
Status	Open
Location	`src/v3/ShopinXToken.sol:12, 53`
Category	Tokenomics / Documentation
CVSS v3.1	0.0
CWE ID	N/A
Remediation Reference	Tokenomics decision - confirm and update docs

Description

The constructor mints the full supply once at deployment. The figure increased from the originally audited 300,000,000 to 1,000,000,000 SPX, which is a tokenomics decision rather than a remediation.

Recommendation

Confirm that the larger supply is the intended tokenomics, and update every document, test, deployment script, and external reference that still states the old figure.

Remediation & Verification

Open and formalised. The supply is now the named constant `INITIAL_SUPPLY` of 1,000,000,000e18 (lines 12 and 53). It does not introduce a vulnerability, but confirm the figure is the intended tokenomics and update every external document and test that still states the older 300,000,000 value.

F-I-10: Unchecked downcasts when writing the vesting struct

Field	Value
ID	I-10
Severity	Informational
Status	Open
Location	`src/v3/ShopinXToken.sol:116-121`
Category	Defensive Coding / Numeric Truncation
CVSS v3.1	0.0
CWE ID	CWE-197 (Numeric Truncation Error)
Remediation Reference	Defensive-coding note - pending

Description

`transferWithVesting` narrows `amount` to `uint128` and `block.timestamp`, `cliffDays`, `stepPercent`, and `stepDays` to `uint32` with unchecked casts. With the current fixed supply and enforced bounds these casts cannot overflow today, but the casts are unchecked and the forge linter flags them as unsafe. `startTime` stored as `uint32` also truncates after the year 2106.

Recommendation

Use a checked conversion (for example OpenZeppelin `SafeCast`) or add explicit `require` bounds when narrowing the values. Document that `startTime` stored as `uint32` is valid until the year 2106.

Remediation & Verification

Open. Raised against this version. There is no impact under the current fixed supply and the enforced bounds, so this is a defensive-coding and long-horizon correctness note rather than a live bug.

Centralisation, Access Control & Privilege Analysis

Overview

The ShopinX Token is a trusted-owner design. The owner holds broad operational powers over locks and vesting, which is inherent to the intended token model and disclosed accordingly (this is the single low-severity centralization notice reported by Aderyn). The blast radius of these powers has been materially reduced in v3: freezes are time-bounded and reversible, transfer locks apply only to the transferred tranche, ownership uses a two-step handover, and the renounce path is disabled. The owner cannot mint new tokens, cannot move or seize tokens held by other addresses, cannot lock any address for longer than four years, and cannot alter or cancel a vesting schedule once created.

On-Chain Roles & Permissions

Role / Address	Permissions	Timelock	Multisig	Risk Level
Owner / Admin	`setLock`, `removeLock`, `transferWithLock`, `transferWithVesting`, ownership transfer	Recommended	Recommended	Medium (bounded by design constraints)

Key Management

Key / Wallet	Purpose	Storage	Backup	Risk Level
Owner Key	Privileged operations	/	/	Reduced with multisig + timelock

Upgradeability Analysis

The contract is not upgradeable. There is no proxy pattern; the deployed bytecode is immutable.

Centralisation Risk Summary

Risk	Severity	Recommendation
Owner holds broad operational powers over locks and vesting	Medium	Assign ownership to a multisig fronted by a timelock
Administrative addresses not yet publicly documented	Low	Publish recipient and owner addresses and their nature

Conclusion

Summary

This contract was reviewed across successive iterations, and the current version represents a substantial improvement over the earlier ones. The higher-severity issues raised previously - the ability to freeze an address permanently, and a transfer lock that applied to a recipient's entire balance rather than only the tokens being distributed - have been resolved: locks are now time-bounded and reversible, and a transfer lock now affects only the amount actually sent. The medium-severity items have also been addressed, partly through code changes and partly through explicit design decisions that are now documented in the source. The items that remain open are low-severity or informational, and are largely product and operational decisions, such as confirming the intended vesting release pattern, deciding whether vesting should be revocable, and publishing the administrative addresses, together with one minor correctness note regarding repeated use of the amount-based lock on the same address. None of the remaining items prevent a safe deployment, provided the operational recommendations are followed.

Before deployment we recommend: addressing the amount-lock overwrite note (L-04), deciding and documenting the burn-while-locked and continuous-release behaviours, setting an explicit `evm_version` for the target chain, considering checked downcasts, and completing the operational items (multisig and timelock on the owner, address publication, and the post-deployment domain separator check).

Positive Observations

- The contract builds on widely used, community-reviewed OpenZeppelin token, ownership, and signature components rather than reimplementing this functionality from scratch.
- The compiler version is pinned to an exact release, and the Solidity 0.8 series provides built-in protection against arithmetic overflow and underflow.
- Ownership uses a two-step handover and the renounce path is disabled, reducing the risk of accidental or irreversible loss of administrative control.
- Administrative locks are bounded in time and reversible, every privileged function validates its inputs, and the public availability view is kept consistent with the rules enforced on actual transfers.

Note

This audit report consists of a security analysis of the **ShopinX Token** smart contract. This analysis did not include economic analysis of the contract's tokenomics. Moreover, we only audited the main contract for the **ShopinX Token** team. Other contracts associated with the project were not audited by our team. We recommend investors do their own research before investing.

Glossary

Abbreviation	Definition
ABI	Application Binary Interface
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DeFi	Decentralised Finance
EOA	Externally Owned Account
ERC	Ethereum Request for Comments (token standards)
EVM	Ethereum Virtual Machine
LoC	Lines of Code
NatSpec	Ethereum Natural Language Specification Format
PoC	Proof of Concept
SAST	Static Application Security Testing
SWC	Smart Contract Weakness Classification

Solidproof.io - Trust Made in Germany

Blockchain Security | Smart Contract Audits | KYC | Marketing

FutureVisions Deutschland UG (haftungsbeschränkt)

Heideland-Ost 34
24976 Handewitt
Germany, Deutschland

Sitz der Gesellschaft / Amtsgericht: Flensburg
HRB 16612 FL
USt-IdNr. DE364783510
Geschäftsführer: Mails Nielsen, Kevin Arens

Solidproof.io is a product of
FutureVisions Deutschland

Please note: You automatically accept our Terms and Conditions and Privacy Policy when using our services.

You can find the current version on our website: https://solidproof.io.


Solid Proofed
Blockchain Security | Smart Contract Audits | KYC
Development | Marketing
MADE IN GERMANY