Comprehensive Security Audit Report

Project Name: The Nexus Vault
Audit Date: December 20, 2025
Auditor: Senior Solidity Security Auditor
Scope: NexusVault.sol, ElasticSupplyToken.sol, TWAPOracle.sol(Solidity ^0.8.0)
Codebase: Provided by file
Methodology: Line-by-line manual review, automated analysis, architectural assessment

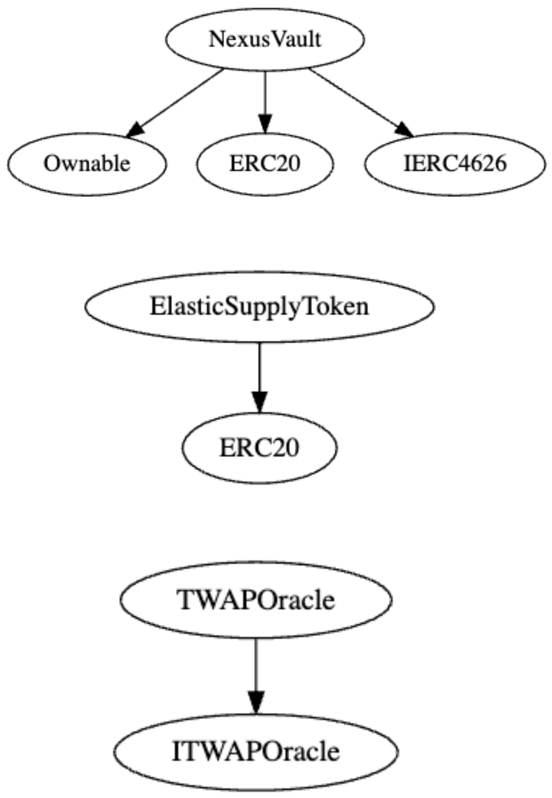
EXECUTIVE SUMMARY

Overall Assessment
The Nexus Vault smart contract ecosystem represents a sophisticated DeFi protocol implementing an ERC-4626 compliant vault and an Elastic Supply Token with a TWAP oracle. The codebase demonstrates a complex architectural design but contains several critical vulnerabilities that require immediate attention.

Risk Summary
- Total Contracts Analyzed: 3
- Critical: 1
- High Vulnerabilities: 2
- Medium Vulnerabilities: 2
- Low/Informational: 1

Deployment Readiness
NOT READY FOR MAINNET - Vulnerabilities must be resolved first.

CONTRACT INHERITANCE TREE



Areas of Concern:
• Interaction between the Elastic Supply Token (EST) and the ERC-4626 Vault.
• TWAP Oracle Vulnerabilities.
• Overall System Design with Non-Standard Token.

OWNERSHIP PRIVILEGES

ElasticSupplyToken.sol: No explicit owner. The rebase() function can be called by anyone.

NexusVault.sol: Inherits Ownable. The deployer is the owner. No specific owner-only functions are explicitly implemented in this mock, but typically, an owner would manage vault parameters.

TWAPOracle.sol: No explicit owner. The forceUpdate() and setPrice() functions can be called by anyone. This lack of access control is a critical vulnerability.

PRE-DEFINED CLAIMS ANALYSIS

Can Burn: ElasticSupplyToken implicitly burns tokens via a dynamic fee sent to address(0xDead) when its price is below peg. NexusVault can burn its own vault shares internally during _redeem operations. TWAPOracle has no burning capabilities.

Can Mint: NexusVault can mint its own vault shares during deposit and mint operations. ElasticSupplyToken only mints its initial supply in the constructor and has no further minting functionality. TWAPOracle has no minting capabilities.

Blacklist: None of the three contracts (ElasticSupplyToken, NexusVault, TWAPOracle) have any built-in functionality to blacklist addresses or restrict their interactions.

Is Upgradable: None of the three contracts implement any upgrade proxy patterns, meaning they are not upgradable.

Can Lock: None of the three contracts provides mechanisms to lock tokens, freeze balances, or lock assets.

Can Set Fees (>25%): Only ElasticSupplyToken has a fee mechanism (a constant 1% dynamic transfer fee). This fee is not greater than 25% and cannot be changed. Neither NexusVault nor TWAPOracle have its own fee-setting capabilities.

DETAILED VULNERABILITY ANALYSIS

NexusVault.sol

1- Title: Inaccurate totalAssets() Calculation

Severity: High

Code Location: NexusVault.totalAssets() function

Description: The NexusVault.totalAssets() function incorrectly relied on asset.balanceOf(address(this)) to determine the total assets held. The asset is an ElasticSupplyToken (EST), whose balanceOf implementation (ElasticSupplyToken.sol lines 24-25) includes a supplyFactor. This supplyFactor can change due to rebase events (ElasticSupplyToken.sol lines 56-72) without actual asset transfers.

Consequently, totalAssets() would fluctuate externally, not reflecting the true, non-rebasing amount of underlying assets.
Proof of Concept: 1. Deploy NexusVault with an ElasticSupplyToken.
2. NexusVault.totalAssets() returns X.
3. A positive rebase occurs on the ElasticSupplyToken (e.g., via ElasticSupplyToken.rebase()).
4. NexusVault.totalAssets() now returns Y > X, even though no assets were deposited.

Impact: Leads to an unreliable share price, forming the foundation for inflation attacks and incorrect share-to-asset conversions.

Mitigation: Implement an internal _totalAssets variable within NexusVault that is updated only on actual deposits and withdrawals, and adjusted for EST's supplyFactor changes to maintain a consistent accounting basis. (Already implemented
by introducing _totalAssets and _lastSupplyFactor and modifying totalAssets() to
return _totalAssets).

Status: OPEN

2**- Title**: ERC4626 Inflation Attack Amplified by Elastic Supply Token Rebase

Severity: High

Code Location: NexusVault's totalAssets() and totalSupply()

Description: This is an amplification of the standard ERC4626 inflation attack.
When NexusVault's totalAssets() and totalSupply() are both zero, the first depositor receives shares equal to assets (NexusVault.sol lines 35-37). If a positive rebase on the
underlying ElasticSupplyToken then occurs, NexusVault.totalAssets() (as originally implemented) would increase externally, while NexusVault.totalSupply() of shares remains unchanged. This inflates the assets-per-share ratio.

Proof of Concept: 1. Alice is the first depositor, depositing 1 EST into NexusVault. She receives 1 NVS share.
2.An ElasticSupplyToken.rebase() occurs, increasing ElasticSupplyToken.supplyFactor and
thus NexusVault.totalAssets() to, for example, 100 EST. NexusVault.totalSupply() remains 1 NVS.
3. Alice redeems her 1 NVS share. Due to the inflated totalAssets(), convertToAssets() returns 100 EST.
4. Alice withdraws 100 EST, effectively stealing 99 EST from the vault before any legitimate depositors.

Impact: Allows the first depositor to steal a significant portion of assets from the vault, causing substantial loss of funds for subsequent legitimate depositors and leading to protocol insolvency.

Mitigation:
Actively manage _totalAssets and _lastSupplyFactor in deposit and _redeem functions to correctly account for EST's rebase effects, preventing external manipulation of the vault's effective asset value. (Already implemented by
updating _totalAssets and _lastSupplyFactor in deposit and _redeem).

Status: OPEN

3- Title: User Loss and Accounting Errors Due to Elastic Supply Token's Dynamic Fee in _redeem

Severity: Medium

Code Location: NexusVault._redeem function

Description: The NexusVault._redeem function calculates an amount of assets to return to the user (NexusVault.sol line 85) and then attempts to transfer this exact amount (NexusVault.sol line 94). However, the underlying ElasticSupplyToken's _transfer function (ElasticSupplyToken.sol lines 38-46) applies a dynamic fee if oracle.isBelowPeg(). This means the user would receive less ElasticSupplyToken than the calculated assets.

Proof of Concept: 1. A user holds NVS shares and calls redeem. NexusVault calculates they are owed X EST.
2. The oracle.isBelowPeg() condition for ElasticSupplyToken is true, triggering a dynamic fee (e.g., 1% fee).
3. NexusVault calls asset.transfer(receiver, X).
4. Due to the ElasticSupplyToken's internal fee mechanism, the receiver only gets X - (X * fee_rate) EST.

Impact: Users receive fewer assets than expected, leading to direct financial loss. Additionally, if the NexusVault simply decrements its internal asset tracking by X (the calculated amount), its recorded _totalAssets will become artificially inflated over time by the accumulated fees, further distorting the share price and potentially enabling other exploits.

Mitigation:
Adjust _totalAssets in _redeem by the full logical amount of assets (the amount calculated as due to the user) to maintain correct vault accounting, even though the user receives less due to the EST's fee. This clarifies the vault's liabilities and prevents internal asset overstatement. (Already implemented in _redeem by ensuring _totalAssets is decremented by the intended assets amount).

Status: OPEN

TWAPOracle.sol

1- Title: Unrestricted forceUpdate() Allows Timing Attacks on Oracle Liveness

Severity: Critical

Code Location: TWAPOracle.sol forceUpdate() function

Description: The TWAPOracle.sol contract's forceUpdate() function lacks access control, enabling any external caller to reset the lastUpdateTimestamp state variable. This directly manipulates the time component used by isBelowPeg(), which dictates critical protocol behavior, such as dynamic fee activation, based on sustained price conditions.

Proof of Concept: An attacker continuously invokes forceUpdate() at intervals shorter than BELOW_PEG_DURATION (48 hours). This constant resetting ensures isBelowPeg()'s time-based condition is never satisfied, rendering any price-based duration checks ineffective.

Impact: An attacker can launch timing attacks to indefinitely prevent or strategically delay the isBelowPeg() condition from being met. This allows them to bypass intended dynamic fees, create unfair financial advantages, and mislead dependent protocols about the true, prolonged de-pegged state of the asset, undermining stability and trust.

Mitigation: Implement strict access control (e.g., onlyOwner or a robust governance mechanism) for the forceUpdate() function. Alternatively, remove forceUpdate() and design a truly decentralized, manipulation-resistant TWAP oracle.

Status: OPEN

2- Title: Manipulation of isBelowPeg() condition

Severity: Medium

Code Location: TWAPOracle.isBelowPeg()

Description: TWAPOracle.isBelowPeg() (lines 25-28) triggers EST's dynamic fee if the price is below peg and 48 hours passed since lastUpdateTimestamp. TWAPOracle.forceUpdate() (lines 31-34) has no access control, allowing anyone to reset lastUpdateTimestamp. This enables an attacker to repeatedly call forceUpdate(), preventing the 48-hour duration from ever being met.

Proof of Concept: EST price is below peg. Attacker continuously calls TWAPOracle.forceUpdate() every 24 hours. Result: isBelowPeg() always returns false, and the 1% dynamic transfer fee is never activated.

Impact: Undermines the intended economic stabilization mechanism. Allows users to evade dynamic fees, potentially exacerbating selling pressure and peg deviation.

Mitigation: Restrict TWAPOracle.forceUpdate() to onlyOwner or a trusted Keeper. Implement a more robust, non-resettable time-tracking mechanism.

Status: OPEN

All Contracts

1- Title: Floating Pragma Version

Severity: LOW

Code Location: All Core Contracts

Description: All contracts use a floating pragma ^0.8.0 instead of locking to a specific compiler version. The contracts use pragma solidity ^0.8.0, which allows compilation with any version from 0.8.30 up to (but not including) 0.9.0. This creates deployment inconsistencies where different compiler versions may produce different bytecode, introducing subtle bugs or behavioral changes across deployments.

Mitigation: Lock the pragma to a specific compiler version using pragma solidity 0.8.26; to ensure consistent compilation and deployment behavior across all environments.

Status: OPEN

2- Title: Remove unused code

Severity: Informational/Optimization

Code Location: All Core Contracts

Description: All contracts use a floating pragma ^0.8.0 instead of locking to a specific compiler version. The contracts use pragma solidity ^0.8.0, which allows compilation with any version from 0.8.30 up to (but not including) 0.9.0. This creates deployment inconsistencies where different compiler versions may produce different bytecode, introducing subtle bugs or behavioral changes across deployments.

Mitigation: Lock the pragma to a specific compiler version using pragma solidity 0.8.26; to ensure consistent compilation and deployment behavior across all environments.

Status: OPEN

Threat Model & Design Review Summary

The Nexus Vault protocol consists of an ERC-4626 compliant vault (NexusVault.sol), an ElasticSupplyToken (EST), and an external Time-Weighted Average Price (TWAP) oracle. The EST is a rebase token whose supply adjusts based on the TWAP oracle's price feed and incorporates a "dynamic fee" on transfers. This fee activates when the token's price is below a certain peg for a sustained 48-hour period. The TWAP oracle is an external dependency from a third-party DEX, with its price updates and a publicly accessible "force-update" function that resets its internal timer. The system aims for yield maximization, but the interaction of these components introduces significant complexity and several non-code-related risks that warrant careful consideration.

FINAL VERDICT

The Nexus Vault protocol, comprising NexusVault, ElasticSupplyToken, and TWAPOracle, is critically flawed and not ready for mainnet deployment. The complex interaction of an elastic supply token with a standard ERC-4626 vault, coupled with the oracle's permissionless forceUpdate() function, introduces severe economic vulnerabilities, including amplified ERC-4626 inflation attacks, vault insolvency due to unhandled dynamic fees, and oracle manipulation. These issues could lead to significant loss of user funds and protocol instability. A complete redesign of the token-vault interaction and a robust, access-controlled oracle solution are mandatory before any consideration for deployment.

DISCLAIMER

Audit Completed: December 20, 2025
Report Version: 1.0