Assure DeFi®

THE VERIFICATION GOLD STANDARD

Security Assessment

Norexa

Date: 23/09/2025

Audit Status: FAIL

Audit Edition: Advanced

ASSURE DEFI®

THE VERIFICATION GOLD STANDARD

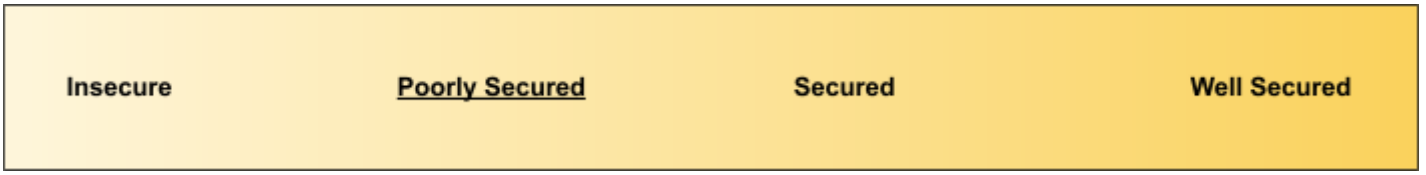
Risk Analysis

Vulnerability summary

Classification	Description
High	High-level vulnerabilities can result in the loss of assets or manipulation of data.
Medium	Medium-level vulnerabilities can be challenging to exploit, but they still have a considerable impact on smart contract execution, such as allowing public access to critical functions.
Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.
Informational	Informational vulnerabilities, code style violations, and informational statements do not affect smart contract execution and can typically be disregarded.

Executive Summary

According to the Assure assessment, the Customer's smart contract is **Poorly Secured.**



Scope

Target Code And Revision

For this audit, we performed research, investigation, and review of the Norexa contracts followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

Target Code And Revision

Project	Assure
Language	Solidity
Codebase	NorexaMarketMakerFactory.sol [SHA256] - b7e757433553744c4d500fa6c66d37045626f96fa64388da0958da5b22e404ce NorexaMarketMakerPoolV2.sol [SHA256] - e4ed0aa99b688b58971a319ed1eef911330c5c8aa9d789ad6db4d2c2ab42ba4b
Audit Methodology	Static, Manual

Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

Category	Item
Code review & Functional Review	• Compiler warnings. • Race conditions and Reentrancy. • Cross-function race conditions. • Possible delays in data delivery. • Oracle calls. • Front running. • Timestamp dependence. • Integer Overflow and Underflow. • DoS with Revert. • DoS with block gas limit. • Methods execution permissions. • Economy model. • Private user data leaks. • Malicious Event log. • Scoping and Declarations. • Uninitialized storage pointers. • Arithmetic accuracy. • Design Logic. • Cross-function race conditions. • Safe Zeppelin module. • Fallback function security. • Overpowered functions / Owner privileges

AUDIT OVERVIEW

1. Unsafe ERC-20 Handling (No SafeERC20 & Unchecked Returns)

Contract: VaultV6, ControllerV3

Function: (multiple) deposit, mint, withdraw, redeem, _withdrawOptimized, claimFees (VaultV6); rebalance, deployToStrategy, emergencyWithdraw, withdrawFromStrategy (ControllerV3)

Issue: Uses raw transfer/transferFrom/approve without verifying return values, non-standard tokens can return false (no revert), enabling share minting without assets or causing silent failures/DoS.

Recommendation: Use OpenZeppelin SafeERC20 (safeTransfer, safeTransferFrom, forceApprove/safeIncreaseAllowance) everywhere and optionally assert balance deltas on deposit to support/guard against fee-on-transfer tokens.

2. EIP-4626 Semantics Bug, withdraw Pays Less Than Requested

Contract: VaultV6

Function: withdraw (and previewWithdraw)

Issue: withdraw(assets) transfers assetsAfterFee (assets minus fee) to user. EIP-4626 requires paying exactly assets and charging fees on top via extra shares burned. previewWithdraw also misleads.

Recommendation: Compute fee=(assets*withdrawalFee)/FEE_PRECISION, set gross=assets+fee. Burn shares covering gross, withdraw gross, transfer exactly assets to user. Mirror logic in previewWithdraw.

3. Public keeper can distort price window - APY forced ~0 (data poisoning / DoS)

Contract: APYOracleV2

Function: updateProtocolPrice

Issue: updateProtocolPrice is publicly callable and lacks any per-protocol rate limit or timestamp normalization. Anyone can rapidly rotate the 7-slot buffer so that ‘week-ago’ and ‘current’ samples are only minutes apart. getHistoricalAPY assumes weekly spacing (no timestamps) and then annualizes, yielding ~0 bps or severely understated APY effectively a cheap data-poisoning/DoS against consumers of the oracle.

Recommendation: Restrict updates to a keeper role and enforce a per-protocol minimum interval (for example ≥1 day). Store timestamps with samples and compute returns using actual elapsed time to prevent cadence manipulation.

4. Zero-share-burn withdrawal - drain over time

Contract: AaveAdapter

Function: AaveAdapter.withdraw(uint256 amount)

Issue: sharesToBurn uses floor division. If amount * totalShares < totalBalance, then sharesToBurn == 0.

The share price equals totalBalance / totalShares. As soon as the share price exceeds 1 aUSDC/share (which it will as yield accrues in Aave), any amount strictly less than the share price produces sharesToBurn == 0.

The function does not enforce sharesToBurn > 0 and the require passes for any user (even with 0 shares) because shares[msg.sender] >= 0 is always true.

The contract then transfers the amount USDC to the caller without burning any shares, creating a hard imbalance that steals value from remaining LPs. Repeating small withdrawals drains the vault.

Recommendation: Use rounding up when converting assets to shares for withdrawals:

```
// ceilDiv(x, y) = (x + y - 1) / y
sharesToBurn = (amount * totalShares + totalBalance - 1) / totalBalance;
require(sharesToBurn > 0, "AaveAdapter: zero shares");
```

Alternatively, switch the API to burn shares as input (like ERC-4626 redeem(shares)), compute assets = shares * totalAssets / totalShares (rounding down on assets), then withdraw exactly assets. This eliminates the “ask for assets, floor shares” pitfall.

Add sanity checks:

require(amount <= maxWithdraw(msg.sender)), where maxWithdraw = shares[msg.sender] * totalBalance / totalShares.

Consider using (or mirroring) OpenZeppelin ERC-4626 conversions to get rounding correct and consistent.

5. Zero-share mint on small deposits - direct user fund loss

Contract: AaveAdapter

Function: AaveAdapter.deposit(uint256 amount)

Issue: When totalShares > 0, share minting uses floor division, so small deposits can mint zero shares.

The function does not enforce sharesOut > 0 and performs the Aave supply before computing shares.

Result: the user USDC is deposited into Aave (increasing the pool aUSDC), but the depositor receives 0 shares a direct user loss, existing shareholders get a free value bump.

Recommendation: Require non-zero shares on mint:

```
sharesOut = (amount * totalShares) / totalBalanceBefore;
require(sharesOut > 0, "AaveAdapter: deposit too small");
```

Or implement min-shares / min-assets slippage parameters (user-provided) and revert if not met.

Consider adopting ERC-4626 math helpers (assets <-> shares), which explicitly define rounding directions and include preview methods so callers can avoid zero-share results.

6. OUSD redemption returns unsupported assets (funds stranded / user under-withdraw)

Contract: OriginAdapter

Function: OriginAdapter.withdraw, swapToUSDCviaUni, getUsdcValue

Issue: withdraw assumes the OUSD vault redeems only into {USDC, USDT, USDS}. If the vault returns other assets (for example, DAI), those tokens remain in the adapter, are never swapped to USDC, and are not counted by getUsdcValue(). Users receive less than their pro-rata value and residual assets accumulate in the contract.

Recommendation: Discover/maintain the exact redeemable asset set (and decimals) from the vault, build minAmounts accordingly, and after redemption swap all non-USDC assets to USDC with per-asset minimums. Include DAI handling. Update valuation to include all assets (or ensure post-op residuals are zero). Use oracles/quoters and enforce user slippage across redemption and swaps.

7. Unbounded AMM slippage on withdrawal swaps (MEV/sandwich loss)

Contract: OriginAdapter

Function: OriginAdapter.swapToUSDCviaUni (used by withdraw)

Issue: Swaps are executed with amountOutMin = 0. Large redemptions can be sandwiched or suffer extreme price impact, returning far less USDC than fair value. Losses are borne directly by the withdrawing user.

Recommendation: Compute amountOutMin from a trusted quote (TWAP/quoter or user-provided minOut) and apply the user's userSlippageBps. Consider routing via a reputable aggregator with built-in slippage controls, and/or support permit-style withdraws that include user-specified minOut.

8. Misaligned slippage controls and incorrect minAmounts math on OUSD redemption

Contract: OriginAdapter

Function: OriginAdapter.withdraw

Issue: minAmounts construction assumes fixed asset order and 6-decimals for non-USDC tokens so this breaks if the vault's asset set/order/decimals differ. The subsequent redemption check compares OUSD burned vs. requested using global slippageBps instead of the user userSlippageBps, and it verifies quantity burned not realized USDC value allowing poor outcomes to pass.

Recommendation: Build minAmounts dynamically against the vault's actual asset set and decimals. Use userSlippageBps consistently. Validate the value received (post-swap USDC or oracle-valued basket) against a user-tolerated minimum, not just OUSD burned.

1. Guardian Authority Confusion in Emergency Keeper Removal

Contract: ControllerV3

Function: emergencyRemoveKeeper

Issue: Dynamically calls guardian() on whatever owner contract is set. Unrelated owner contracts exposing guardian() can unintentionally gain keeper-removal rights.

Recommendation: Store an explicit guardian set by owner, or verify owner is a known TimelockController (for example by interface flag) before trusting its guardian() or otherwise ignore.

2. Decimals-Dependent Hardcoded Thresholds

Contract: VaultV6

Function: Constants used by _rebalanceIdleFunds, first-deposit checks

Issue: MIN_REBALANCE_AMOUNT = 10_000e6 and INITIAL_DEPOSIT_MINIMUM = 1000 assume 6-decimals (USDC). Using other assets breaks thresholds (DoS or weak protection).

Recommendation: Scale thresholds by decimals() or make them owner-configurable with bounds. Recompute at deploy or on asset change.

3. Approval Pattern Can Brick Ops on Non-Standard Tokens

Contract: ControllerV3

Function: rebalance, deployToStrategy

Issue: Uses approve(amount) directly. Tokens like USDT require zeroing allowance first or may return false, causing DoS without detection.

Recommendation: Use SafeERC20.forceApprove (or set to 0 then to amount). Handle return values and errors, prefer exact-amount allowances per call.

4. Unauthorized Guardian Access in Emergency Functions

Contract: ControllerV3

Function: emergencyRemoveKeeper() function

Issue: The emergencyRemoveKeeper() function allows a guardian to remove keepers without proper validation. The guardian address is obtained through an external call to TimelockController(owner).guardian(), but there's no verification that the owner is actually a valid TimelockController or that the guardian role is legitimate.

Recommendation: Implement proper guardian validation.

Add a whitelist of approved TimelockController contracts.

Consider implementing a time delay for guardian actions.

5. First Depositor Inflation Attack Mitigation Insufficient

Contract: VaultV6.sol

Function: Deposit functions

Issue: While there's a minimum deposit protection, the INITIAL_DEPOSIT_MINIMUM = 1000 (0.001 USDC) is too low to effectively prevent inflation attacks on vault shares.

Recommendation: Increase minimum first deposit to meaningful amount (like $1000)

Consider burning first deposit shares to prevent manipulation

Implement share price bounds checking

6. Batch interval guard bypass via single-protocol updater

Contract: APYOracleV2

Function: updateDailyPrices, updateProtocolPrice

Issue: updateDailyPrices enforces UPDATE_INTERVAL, but updateProtocolPrice has no interval checks, allowing attackers to bypass the intended cadence control and spam updates for specific vaults.

Recommendation: Apply the same (or stricter) per-protocol interval guard inside updateProtocolPrice, or remove public access and route all updates through a keeper-gated, rate-limited path.

7. APY calculation assumes fixed weekly spacing (no time normalization)

Contract: APYOracleV2

Function: getHistoricalAPY

Issue: The function always compares adjacent indices and multiplies by 52 without considering real time between samples. Missed days or bursts systematically misprice APY (over- or under-estimation) even without an attacker.

Recommendation: Store (price, timestamp) per slot and annualize by actual elapsed seconds (for ex, APY ≈ [(current/old)^(SECONDS_PER_YEAR/dt) – 1] in bps). Consider EMA/TWAR to reduce cadence sensitivity.

8. Deposit division-by-zero cause permanent DoS when strategy value is zero

Contract: Compounder, Morpho, Vesper, Yearn

Function: CompoundAdapter.deposit, MorphoAdapter.deposit, VesperAdapter.deposit, YearnAdapter.deposit

Issue: adapters compute sharesOut = amount * totalShares / totalValueBefore. If totalShares > 0 but totalValueBefore == 0 (like upstream loss/paused vault), division by zero reverts, blocking all new deposits and creating an availability DoS.

Recommendation: Add a guard: if totalShares > 0 && totalValueBefore == 0, revert with a clear error ("strategy value zero; deposits paused") and expose a governance recovery path (for example write-down/reset). Optionally add Pausable to gate deposits during incidents.

1. Residual Allowances Left on Strategies

Contract: ControllerV3

Function: rebalance, deployToStrategy

Issue: After deposits, leftover allowance may remain to strategy, enabling unintended future pulls if balances later appear.

Recommendation: Reset allowance to 0 after successful deposit (or use scoped exact-amount forceApprove before, then clear).

2. Preview vs Execution Drift with Fee-On-Transfer Tokens

Contract: VaultV6

Function: previewDeposit, previewMint, deposit, mint

Issue: Previews assume 1:1 transfers. Fee-on-transfer tokens cause fewer assets received than assumed to share/accounting mismatch and user confusion..

Recommendation: Either explicitly reject fee-on-transfer assets, or compute actual received via balance-delta and base shares on that. Adjust previews to conservative estimates or document unsupported tokens.

3. Vault mapping accepts unsupported vaults - permanent APY=0 for those adapters

Contract: APYOracleV2

Function: setAdapterVaultMapping, _getCurrentPrice, updateProtocolPrice

Issue: Owner can map an adapter to any vault, but _getCurrentPrice only supports three hardcoded vaults.

Unsupported vaults return price=0 causing updates to revert (“InvalidPrice”), so initialization never completes and APY remains 0 silent misconfiguration DoS.

Recommendation: Validate vaults against a supported registry (or implement pluggable pricing strategies per vault type). Revert on unsupported mappings with a clear error.

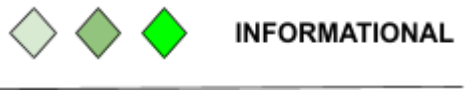
4. Adapter enumeration returns incomplete set

Contract: APYOracleV2

Function: getRegisteredAdapters

Issue: The function only returns adapters for five hardcoded names. Legitimately registered adapters with other names are omitted, misleading off-chain consumers/UIs.

Recommendation: Maintain a dynamic list (or enumerable set) of all registered adapters on register/unregister and return that list directly.



No informational issues were found.

Technical Findings Summary

Findings

Vulnerability Level		Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
	HIGH	8					
	MEDIUM	8					
	LOW	4					
	INFORMATIONAL	0					

Assessment Results

Score Results

Review	Score
Global Score	**65/100**
Assure KYC	Not completed
Audit Score	65/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below. The Global Score is a combination of the evaluations obtained between having or not having KYC and the type of contract audited together with its manual audit.

Audit FAIL

Following our comprehensive security audit of the token contract for the Norexa project, the project did not fulfill the necessary criteria required to pass the security audit.

Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocating for the Project, and users relying on this report should not consider this as having any merit for financial adNorexa in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment adNorexa, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided ‘as is, and Assure Defi is under no covenant to audit completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment serNorexas provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any serNorexas, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The serNorexas may access, and depend upon, multiple layers of third parties.