AstraSec

ProtocolVault
Security Audit Report

January 24, 2026

Contents

1 Introduction

1.1 About ProtocolVault

ProtocolVault is an ERC4626-compliant yield vault that accepts USDe deposits and allocates capital to target tokens to generate returns. The protocol provides users with exposure to target assets through a standardized vault interface, enabling automated asset management and liquidity provision. The vault operates with role-based access control for operational management and includes safeguards for user protection.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/stratosphere-network/ProtocolVault.git
- CommitID: 1b636ca

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/stratosphere-network/ProtocolVault.git
- CommitID: 67e4b52

Please note this audit focuses on the security of the contracts themselves. The correctness and reasoning of the economic model itself are not within the scope of this audit.

1.3 Revision History

Version	Date	Description
v1.0	January 19, 2026	Initial Audit

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the ProtocolVault project. Throughout this audit, we identified a total of 2 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	–	–	–	–
High	2	1	–	1
Medium	–	–	–	–
Low	–	–	–	–
Informational	–	–	–	–
Undetermined	–	–	–	–

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and re-mediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

3.3.1 [H-1] Possible Price Manipulation in updateTokenAPrice()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
ProtocolVault.sol	Business Logic	High	Medium	Mitigated

The updateTokenAPrice() function relies on a short 300-second (5-minute) TWAP fetched from the Island contract, which is highly vulnerable to manipulation. Attackers can exploit large trades, or sandwich attacks to significantly skew the recent pool price, distorting the TWAP and allowing temporary inflation or deflation of tokenAPrice. Since this price directly determines the vault's total value (getVaultValue()), share prices, redemption amounts, and overall protocol economics, manipulated prices enable profitable attacks via timed deposits, redemptions.

Additionally, the price conversion logic assumes tokenA is always token0 in the pool (price = token1 / token0). If tokenA is token1, the computed price becomes the reciprocal (inverted), producing severely incorrect tokenAPrice values that break vault accounting and redemption mechanics.

Furthermore, the raw integer arithmetic (sqrtPriceSquared * 1e18) / Q96Squared is susceptible to precision loss from truncation during division and risks intermediate overflow at extreme price levels.

ProtocolVault-main - ProtocolVault.sol

```
function updateTokenAPrice() external onlyRole(PRICE_FEED_ROLE) {
    // Call Island contract to get 5-minute TWAP (300 seconds)
    IIsland island = IIsland(ISLAND_CONTRACT);
    uint160 avgSqrtPriceX96 = island.getAvgPrice(300);

    require(avgSqrtPriceX96 > 0, "Invalid price from Island contract");

    // Convert sqrtPriceX96 to regular price with 18 decimals
    uint256 sqrtPriceX96 = uint256(avgSqrtPriceX96);
    uint256 Q96 = uint256(2**96);

    // Calculate sqrtPriceX96^2
    uint256 sqrtPriceSquared = sqrtPriceX96 * sqrtPriceX96;

    // Calculate Q96^2 = 2^192
    uint256 Q96Squared = Q96 * Q96;

    // Calculate: (sqrtPriceSquared * 1e18) / Q96Squared
    // Multiply by 1e18 first to maintain precision
    uint256 price = (sqrtPriceSquared * 1e18) / Q96Squared;

    require(price > 0, "Price conversion failed");

    tokenAPrice = price;
    emit PriceUpdated(tokenA, price, block.timestamp);
}
```

Remediation (1) extend the TWAP window to reduce manipulation risk, (2) detect token order by comparing token addresses and inverting the price calculation when tokenA is token1, (3) add overflow checks and using safer arithmetic operations.

3.3.2 [H-2] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
ProtocolVault.sol	Security	High	Medium	Acknowledged

The ProtocolVault relies on multiple privileged accounts that possess extensive control over critical operations, introducing notable centralization risks. These accounts, assigned distinct roles, can unilaterally influence the protocol's functionality and integrity. Below are key examples of privileged functions and their associated roles:

- ➔ **Contract Updater Role (DEFAULT_ADMIN_ROLE):** Authorizes contract upgrades via _authorizeUpgrade(), allowing changes to core protocol logic (deposits, withdrawals, fees, accounting, swap mechanisms).
- ➔ **Admin Role (DEFAULT_ADMIN_ROLE):** Manages vault configuration (fee address, mint whitelists, router address), executes rebalance() to charge management and performance fees, pauses/unpauses vaults, triggers emergencyWithdraw() to extract any ERC20 tokens, and can mint shares using adminMintShares(). These operations can modify how user assets are invested, valued, or made available for withdrawal.
- ➔ **Operator Role (OPERATOR_ROLE):** Controls asset allocation by executing swaps between USDe and TokenA via swapToToken() and swapToStable(). Controls where user capital is allocated and can affect returns through swap timing and execution.
- ➔ **Price Feed Manager Role (PRICE_FEED_ROLE):** Updates vault valuation through updateTokenAPrice().

ProtocolVault-main - ProtocolVault.sol

```
140 function adminMintShares(
141     address to,
142     uint256 shares,
143     string memory reason
144 ) external onlyRole(DEFAULT_ADMIN_ROLE) {
145     require(to != address(0), "Invalid address");
146     require(shares > 0, "Shares must be > 0");
147     require(whitelistedMintAddresses[to], "Address not whitelisted for minting");
148
149     _mint(to, shares);
150
151     emit SharesMinted(to, shares, reason);
152 }
153
```

ProtocolVault-main - ProtocolVault.sol

```
534 function _authorizeUpgrade(address newImplementation) internal override onlyRole(DEFAULT_ADMIN_ROLE) {}
```

ProtocolVault-main - ProtocolVault.sol

```
218 function emergencyWithdraw(
219     address token,
220     uint256 amount,
221     address receiver
222 ) external onlyRole(DEFAULT_ADMIN_ROLE) nonReentrant {
223     require(token != address(0), "Invalid token");
224     require(receiver != address(0), "Invalid receiver");
225
226     uint256 balance = IERC20(token).balanceOf(address(this));
227     uint256 withdrawAmount = amount == 0 ? balance : amount;
228     require(withdrawAmount > 0, "No balance to withdraw");
229     require(withdrawAmount <= balance, "Insufficient balance");
230
231     SafeERC20.safeTransfer(IERC20(token), receiver, withdrawAmount);
232
233     emit EmergencyWithdraw(receiver, token, withdrawAmount);
234 }
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.
Response By Team This issue has been confirmed by the team.

4 Appendix

4.1 About AstraSec

AstraSec is a premier blockchain security firm dedicated to delivering high-quality auditing services for blockchain-based protocols. Composed of veteran security researchers with extensive experience across the DeFi landscape, our team maintains an unwavering commitment to excellence and precision. AstraSec's comprehensive methodology and deep understanding of blockchain architecture enable us to ensure protocol resilience, making us a trusted partner for our clients.

4.2 Disclaimer

The content of this audit report is for informational purposes only and does not constitute legal, financial, or investment advice. The findings, views, and conclusions presented herein are based on the code and documentation provided at the specific time of the audit and may be subject to risks and uncertainties not detected during the assessment.

While AstraSec exerts every effort to ensure the accuracy and completeness of this report, the information is provided on an "as is" basis. We assume no responsibility for any errors, omissions, or inaccuracies. Users should conduct their own independent due diligence and consult with professional advisors before making any investment or deployment decisions. AstraSec shall not be held liable for any direct or indirect losses, damages, or consequences arising from reliance on this report.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://x.com/AstraSecAI