yAudit USDAF PR#2 Review

Review Resources:

- USDAF PR#2

Auditors:

- Panda
- Fedebianu

Table of Contents

Review Summary

USDAF PR review

This review covers pull request #2 to the USDAF codebase that introduces a new oracle and adjusts the oracle stallness checks.

The PR of the USDAF Repo was reviewed over a two-day period. Two auditors performed the code review between July 7th and July 8th, 2025. The review was limited to the latest pull request commit 0a39d60615e3b394ca88fa07058f6a3fd0f462f4.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
contracts
  ├── script
  │   └── DeployUSDaf.s.sol
  └── src
      └── PriceFeeds
          └── USDaf
              ├── BaseOracle.sol
              ├── ERC4626Oracle.sol
              └── StyBoldOracle.sol
```

After the findings were presented to the USDAF team, fixes were made and included in the existing PR or inside new PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, USDAF and users of the contracts agree to use the code at their own risk.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Undetermined
    - Findings whose impact could not be fully assessed within the time constraints of the engagement. These issues may range from low to critical severity, and while their exact consequences remain uncertain, they present enough potential risk to warrant attention and remediation.
- Gas savings
    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

Critical Findings

None

High Findings

None

Medium Findings

None

Low Findings

1. Low - Narrow staleness tolerance in BTCPriceFeed

While most staleness thresholds have been updated to account for potential delays in Chainlink's heartbeat, the BTC/USD feed used internally in the BTCPriceFeed contract still references a one-hour period.

Technical Details

BTCPriceFeed.sol#L11

Impact

Low. A stale feed would only disable the redemption price toggle and wouldn't cause a branch shutdown.

Recommendation

Consider updating this value to increase the staleness tolerance, consistent with other cases in the codebase.

Developer Response

Fixed in f07b514.

Gas Saving Findings

None

Informational Findings

1. Informational - `SusdsOracle` should use `USDS` CL price feed

Technical Details

`SusdsOracle` is currently configured to use the `DAI` Chainlink price feed at address `0xAed0c38402a5d19df6E4c03F4E2DceD6e29c1ee9`. However, the correct Chainlink price feed for `USDS` is at address `0xfF30586cD0F29eD462364C7e81375FC0C71219b1`. Even if Sky lets convert `DAI` for `USDS` at a 1:1 exchange rate, using the `USDS` price feed will result in a more robust and accurate price reference for the `USDS` token.

Impact

Informational.

Recommendation

Update the Chainlink price feed address for `USDS` to `0xfF30586cD0F29eD462364C7e81375FC0C71219b1`.

Developer Response

Fixed: [81209aea1b9c93e048a38fa1811b454feb8b086a]

2. Informational - Specify all the assumptions in `ERC4626Oracle.latestRoundData()`

Technical Details

In the code, there is a comment and logic that assumes the ERC4626 vault token `TOKEN` has 18 decimals:

```
// assumes that `TOKEN` has 18 decimals
answer = answer * int256(TOKEN.convertToAssets(_WAD)) / int256(_WAD);
```

However, this approach works correctly only if both the vault token `TOKEN` and its underlying asset `TOKEN.asset()` have 18 decimals. If the underlying asset uses a different number of decimal places, the calculation will be inaccurate, resulting in incorrect price reporting. This unspecified assumption does not directly introduce a vulnerability but may cause future development errors if the contract is ever used with tokens or assets that do not both have 18 decimals.

Impact

Informational.

Recommendation

Make the code generic by dynamically fetching the decimals of both the vault token and its underlying asset. Use the correct scaling factors in calculations to ensure accuracy regardless of the decimals used by the tokens involved.

```
uint8 assetDecimals = IERC20Metadata(TOKEN.asset()).decimals();
answer = answer * int256(TOKEN.convertToAssets(10 ** TOKEN.decimals())) /
```

```
int256(10 ** assetDecimals);
```

Or, at least, specify all the assumptions in the comment:

```
// assumes that `TOKEN` and `TOKEN.asset()` have 18 decimals
answer = answer * int256(TOKEN.convertToAssets(_WAD)) / int256(_WAD);
```

Developer Response

won't fix

3. Informational - Hardcoded Assumption on BOLD Solvency

Technical Details

The `StyBoldOracle` contract inherits from `ERC4626Oracle`. It calculates the price of `st-yBOLD` by chaining `convertToAssets()` calls: first on the staked token (`st-yBOLD`), then on its underlying non-staked token (`yBOLD`). However, it assumes that the value of `yBOLD` is always equal to `1 USD`, pegged.

This assumption is implicitly hardcoded into the logic, as no actual price feed is used to verify the value of BOLD—only its internal accounting via `convertToAssets()` is referenced, since USDAF shares the same codebase and pricing mechanism, any depeg or critical bug in BOLD (e.g., manipulation or misconfiguration of `convertToAssets()`) would cause incorrect pricing in USDAF as well, regardless of USDAF's reserves or status.

Impact

Informational. If BOLD depegs or suffers from a critical logic bug (e.g., erroneous asset conversion), then the pricing mechanism of both `st-yBOLD` and any derivatives like USDAF that depend on it would also reflect incorrect prices. This could lead to cascading failures across protocols using these oracles—such as undercollateralized loans, invalid redemptions, or mispriced swaps—without any on-chain indicator or fallback to prevent it.

Recommendation

Document this assumption explicitly in both code and technical docs to inform integrators and downstream protocol developers.

Developer Response

won't fix