Pegasus permissioned launch

Security Review

Cantina Managed review by:
Xmxanuel, Lead Security Researcher
Deadrosesxyz, Security Researcher
Jonatas Martins, Associate Security Researcher

May 21, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Usual is a Stablecoin DeFi protocol that redistributes control and redefines value sharing. It empowers users by aligning their interests with the platform's success.

$USD0 is a USUAL native stablecoin with real-time transparency of reserves, fully collateralized by US Treasury Bills. This eliminates fractional reserve risks and protects against the bankruptcy risks of fiat-backed stablecoins.

$USD0 can be locked into $USD0++, a liquid 4-year bond backed 1:1, offering users the alpha-yield distributed as points and ensuring at least the native yield of their collateral. This provides enhanced stability and attractive returns for holders.

From Apr 29th to May 3rd the Cantina team conducted a review of pegasus-permissioned-launch on commit hash d0b87f18. The team identified a total of **26** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	6	2	4
Low Risk	7	7	0
Gas Optimizations	1	0	1
Informational	11	9	2
Total	**26**	**19**	**7**

3 Findings

3.1 High Risk

3.1.1 If `cbr` activated: incorrect `cbrcoef` formular results in too high redeem amounts and protocol insolvency

Severity: High Risk

Context: *(No context files were provided by the reviewer)*

Description: In the event that the total value of the `RWA` tokens locked in the protocol is less than the total supply of USD0 tokens, a 1:1 exchange rate cannot be supported anymore, as it could lead to a bank run.

Therefore, a counter-bank run (CBR) mechanism needs to be activated.

A `cbr coefficient` (cbrCoef) would be calculated to reflect the real collateral value, which can be utilized in a redemption process.

However, this is currently done incorrectly.

Incorrect Current Formula:

```
uint256 cbrCoef_ = Math.mulDiv(
    wadTotalRwaValueInUsd + wadUsdBalanceInInsurance,
    SCALAR_ONE,
    totalUsdSupply,
    Math.Rounding.Ceil
);
```

The problem is related to the `insurance fund`.

Additionally, the system has an insurance fund (`wadUsdBalanceInInsurance`) in `USD0` tokens. These tokens would be burned as a last resort to improve the exchange ratio.

It is not required to burn the tokens immediately; the insurance fund can be viewed as a commitment to not redeem the tokens. If prices improve again, the `cbrCoef` could be adjusted accordingly. If the price remains the same, a burn would be required.

Therefore, the insurance fund needs to be considered as burned in the `cbrCoef` calculation to obtain the improved `cbrCoef`.

However, this is currently done incorrectly. Burning the insurance fund would result in a decrease in the `totalSupply` and therefore needs to be subtracted in the formular.

Correct Formula:

```
uint256 cbrCoef_ = Math.mulDiv(
    wadTotalRwaValueInUsd ,
    SCALAR_ONE,
    totalUsdSupply - wadUsdBalanceInInsurance,
    Math.Rounding.Ceil
);
```

Proof of concept: Just one example to illustrate with 100m usd0 and a price drop/loss of 5%:

total usd0 supply	100m
total rwa value locked	95m
insurance fund	1m

In their current version it would result in a `coef = (95 + 1)/100) = 0.96`

However the correct coef should be: `(95/100 - 1)= 0.959595959`.

For simplification, we don't consider a price improvement and the insurance fund needs to be burned.

After the burn the total supply would be `99m` usd0 and `95m` collateral value.

The coef of 0.96 would lead to: `99m * 0.96 = 95.04m`

Only 95m would be available. The protocol would be insolvent with 40k USD0 left and no corresponding collateral.

Recommendation: The `wadUsdBalanceInInsurance` needs to be `subtracted` from the `totalUsdSupply` in the `cbrCoef` formular.

Usual: Fixed in commit 237f6cda. `cbrCoef` is passed as a parameter to the `activateCBR` function.

Cantina Managed: Fixed.

3.2 Medium Risk

3.2.1 No sanity check for `oracle.latestrounddata` return values

Severity: Medium Risk

Context: ClassicalOracle.sol#L80

Description: Currently, the only validation of the return values from `lastRoundData` is if the answer is smaller or equal than zero.

Recommendation: When fetching a price from a chainlink oracle consider the following checks:

- `oracle.latestRoundData`

```
function latestRoundData() external view
    returns (
        uint80 roundId,
        int256 answer,
        uint256 startedAt,
        uint256 updatedAt,
        uint80 answeredInRound
    )
```

Sanity Checks:

- Consider basic sanity checks:

```
require(answer >= 0);
require(updatedAt != 0);
require(updatedAt <= block.timestamp);
```

- Consider a check for staleness with a reasonable `timeout`:

```
require(block.timestamp - updatedAt < timeout)
```

- Consider storing the last good price as a potential fallback mechanism: `answeredInRound` is deprecated in the lastest Chainlink version. If for some reason one rwa token uses an older price aggregator. Consider the following check:

```
require(answeredInRound >= roundId, "Stale price");
```

Usual: Fixed in commit bbaf4bf. Added a timeout check.

Cantina Managed: Fixed.

3.2.2 Not activating or too late activation of `cbr` mechanism results in too high redeem amounts

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: In case the total value of the rwa tokens locked in the protocol is less than the total supply of USD0 tokens, the `cbr` mechanism needs to be activated, immediately.

If not activated or too late activated `user` would still receive a 1:1 exchange rate in redeem. This would be to the cost of not redeeming `usd0` holders.

Once the `cbr` is finally activated, the previously inflated redemption rate will result in a lower `cbrCoef` to balance out the protocol loss.

Not activating the `CBR` at all can lead to protocol insolvency in the worst case.

Recommendation: The entire system needs to be monitored carefully with quick response time. Consider potential alternative designs, where the `cbrCoef` could be updated or activated by anyone.

Usual: Acknowledged. Our off-chain monitoring system monitors the protocol in real time and updates the CBR accordingly.

Cantina Managed: Acknowledged.

3.2.3 Users could be blocked to redeem if not on the allowlist for all `rwa` tokens

Severity: Medium Risk

Context: DaoCollateral.sol#L570

Description: For receiving and transferring `rwa` tokens the user needs to be on the allowlist of a `rwa` token.

This typically involves a KYC process. However, it is highly likely that for some users that's not possible due to legal restrictions in their country of residency, etc...

The `DAOCollateral` contract can support up to 10 different rwa collateral types. There is no guarantee to choose a specific `rwa` token in the `redeem` step. Users might be required to redeem a different `rwa` token they initially swapped.

However, if getting on the allowlist is not possible for a specific user. It would be not possible to `redeem`.

Recommendation: This risk should be documented and users need to be aware. The contract could offer a `view` function to verify if the `user` is on the `allowlist` for all rwa tokens.

If not enforced on the contract level the app could inform the user about it before executing a `swap` transaction.

Usual: Acknowledged. We are launching with a single RWA provider during the permissioned launch, which makes this a non-issue for us until we add new ones, which we would vet carefully to prevent this from occurring.

Cantina Managed: Resolved.

3.2.4 User could sandwich oracle updates and steal treasury funds

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: When swapping/redeeming `USD0`, the received amounts depend directly on the latest reported Chainlink price. Considering an oracle deviation of 2%, would allow user to make a guaranteed profit of `2% - redeemFee` upon every such change of oracle price.

Example scenario:

1. Current reported RWA price is 1000 USD. The price with which it will be updated is 1020 USD.
2. User front-runs the oracle price change and redeems 1000 USD0 for 1 RWA.
3. Price gets updated to 1020 USD.
4. User then swaps the RWA back for 1020 USD0.

The example scenario did not include a `redeemFee` for simplicity, but as long as the `redeemFee` is less than the oracle's deviation, the attack will be profitable.

Recommendation: Keep the `redeemFee` close to the oracle's deviation. Do not allow for both swaps and redeems at the same time - allow for only one at a time.

Usual: Acknowledged. Our chosen `redeemFee` already protects against this attack vector, as sandwiching the oracle updates would not be profitable for the attacker.

Any attacker additionally would require to be allow-listed by one of our RWA providers to be able to attempt an attack. Our RWA providers require a KYC & vetting procedure to be allow-listed, which further reduces the attack surface.

As an additional security measure, the oracle price changes from our RWA providers are broadcast on protected RPCs to further reduce the occurrence of front-running

Cantina Managed: Acknowledged.

3.2.5 Allow users to set `minamountout` on swaps and redeems

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: When swapping/redeeming within `DAOCollateral`, the amount of tokens a user will receive directly depends on the latest price reported by the oracle (and in case of redeems - the current `CBRCoef`). As these values can change before a user's transaction executes, or it can simply remain pending for long enough time, this could force users to take unfair swaps/redeems, which they wouldn't otherwise do.

Recommendation: Allow users to input `minAmountOut` that they're willing to receive.

Usual: Fixed in commit 237f6cd.

Cantina Managed: Fixed.

3.2.6 Oracle will return wrong price if it goes out of chainlink's `minanswer`/`maxanswer`

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: Chainlink oracles have a `minAnswer-maxAnswer` price range in which they report prices. If an asset's price goes out of this range, it will continue to report the price which the asset has crossed.

For example, if a RWA's price drops below its set `minAnswer`, Chainlink will continue to report `minAnswer` as its price, allowing for users to swap it for more USD0 than what it is really worth.

Recommendation: Check if the provided price by the oracle equals to `minPrice/maxPrice` and revert in such case.

Usual: Acknowledged. Our RWA providers' oracles did not implement this specific Chainlink function.

Cantina Managed: Acknowledged.

3.3 Low Risk

3.3.1 Only powerful admin key can add new members to `usd0` allowlist no roleadmins enabled

Severity: Low Risk

Context: RegistryAccess.sol#L14

Description: The `RegistryAccess` contract uses the `AccessControlDefaultAdminRulesUpgradeable` from OpenZeppelin.

In the current implementation, the internal `_setRoleAdmin` function is not exposed. This means all roles can be only `granted` by the admin. These can turn out to be very impractical. The `admin` role is very powerful an can stop the protocol.

On the other hand adding new allowlist members to the `USD0` might happen very often and should require less signatures.

Recommendation: It is possible to manage different roles with a Safe Multi-Sig or Llama.

However, if already enforced at the protocol level, we recommend exposing the `_setRoleAdmin` function to allow a different `roleAdmin` for the `ALLOWLISTED` role and others.

Usual: Fixed in commit 237f6c.

Cantina Managed: Fixed.

3.3.2 Treasury can collect `fees` with no corresponding locked collateral because collateral has been returned to the `redeem` user

Severity: Low Risk

Context: DaoCollateral.sol#L517

Description: In the current design some of the collected `treasury fees` can have no corresponding collateral in the `DAOCollateral`.

Because the corresponding collateral has been already returned to the user in the `redeem`. This only happens if the `rwaToken` has a lower precision like 10^6. (USDC).

The problem relies in the fact, that fees are collected in the higher precision of 10^18 and are not normalized for the lower precision collateral.

Proof of concept: The testcase illustrates the edge case of the last `redeem` where afterwards no collateral is left in the `DAOCollateral` but some fee amount is left in the treasury.

However, this small `dust` error can happen in each redemption depending on the passed amount.

```
contract FeeTest is SetupTest {
    function wadTokenAmountForWadPrice(
        uint256 wadStableAmount,
        uint256 wadPrice,
        uint256 tokenDecimals
    ) internal pure returns (uint256) {
        return Math.mulDiv(wadStableAmount, 10 ** tokenDecimals, wadPrice, Math.Rounding.Floor);
    }

    function testMissingFeeCollateral() public {
        // system state:
        uint256 usd0Amount = 99_999_999_999_999; // total minted usd0
        uint256 rwaTokenAmount = 99; // rwa tokens in daoCollateral
        uint8 rwaDecimals = 6;
        uint256 price = 1e18;
        uint256 fee = 1; // 1 bps

        // simulate redeem
        uint256 stableAmount = usd0Amount;
        // 1. calculate stable fee
        uint256 stableFee = Math.mulDiv(usd0Amount, fee, 10_000, Math.Rounding.Floor);

        // 2. stable fee would be transferred to treasury
        uint256 treasuryUSD0Balance = stableFee;

        // 3. calculate burn amount
        uint256 burnedStable = stableAmount - stableFee;

        // 4. calcuate returned Collateral
        uint256 returnedCollateral = wadTokenAmountForWadPrice(burnedStable, price, rwaDecimals);

        // all rwa tokens are redeemed from DAOCollateral
        // after the redeem no collateral left in DAOCollateral
        assertEq(rwaTokenAmount, returnedCollateral);

        // PROBLEM: fees in the treasury have corresponding collateral
        assertTrue(treasuryUSD0Balance > 0);
        // usd0 treasury balance without corresponding collateral
        assertEq(treasuryUSD0Balance, 9_999_999_999);

        // This happens because of the following
        assertEq(
            wadTokenAmountForWadPrice(burnedStable, price, 6),
            wadTokenAmountForWadPrice(burnedStable + stableFee, price, 6)
        );
    }
}
```

Recommendation: Normalize the `stableFee` in `_tranferFree` before sending it to the treasury.

1. Convert the `stableFee` to precision of the `rwaToken` (This will remove the double counted `usd0` amounts).
2. Convert the stable fee back to the precision of the 10^18 `usd0`.

3. The existing `stableFee > 0` check before transfer.

Although the problem is only about the dust amount, we recommend fix it to build a clean, elegant system.

Usual: Fixed in commit 89da9d3.

Cantina Managed: Fixed.

3.3.3 Incorrect rounding in `cbrcoef` use `math.rounding.floor` instead of `math.rounding.ceil`

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: Currently the rounding in the `cbrCoef` formular is in favor of the users with `Math.Rounding.Ceil` but it should be `Math.Rounding.Floor`.

Currently, you would give away too many `rwaTokens` to the users which would hurt the last redeem `usd0` holder.

Invariant: After the `cbrCoef_` is activated it should be possible to redeem all `rwa` tokens. Assuming no price changes after the activation. This invariant is currently broken.

Proof of concept:

```
function testRounding() public {
    // scenario:
    // 100 usd0 tokens minted
    // 100 rwa token in daoCollateral
    // price = 1 * 1e18 - 1
    // assuming a wadUsdBalanceInInsurance = 0

    uint256 wadTotalRwaValueInUsd = 100 * 1e18 - 1;
    uint256 totalSupply = 100 * 1e18;
    uint256 price = 1 * 1e18 - 1;
    uint256 cbrCoef_ =
        Math.mulDiv(wadTotalRwaValueInUsd, 1e18, totalSupply, Math.Rounding.Ceil);
        // cbrCoef_ will be 1e18 or 1e18-1 depending on Math.Rounding.Ceiling or Math.Rounding.Floor
    // assertEq(cbrCoef_, 1 * 1e18);

    // redeem calculations to get rwa tokens
    uint256 amount = wadTokenAmountForWadPrice(100 * 1e18, price, 18);
    uint256 amountInToken = Math.mulDiv(amount, cbrCoef_, 1e18, Math.Rounding.Floor);

    // this will revert with Math.Rounding.Ceil but not with Math.Rounding.Floor
    // the protocol can't give away more tokens
    uint256 rwaTokenAvailable = 100 * 1e18;
    assertTrue(amountInToken <= rwaTokenAvailable);
}
```

Recommendation: Replace `Math.Rounding.Ceil` with `Math.Rounding.Floor` in the coeff calculation.

Usual: Fixed in commit 237f6c. The `cbrCoef` is not calculated in the contract anymore.

Cantina Managed: Fixed.

3.3.4 Follow chainlink best practices and use `proxy` instead of `priceaggregator` directly

Severity: Low Risk

Context: ClassicalOracle.sol#L80

Description: Chainlink recommends using the `proxy` and not the `priceAggregator` directly as a best practice.

Recommendation: Follow the mentioned best practices from Chainlink:

> You can call the latestRoundData() function directly on the aggregator, but it is a best practice to use the proxy instead so that changes to the aggregator do not affect your application. Similar to the proxy contract, the aggregator contract has a latestAnswer variable, owner address, latestTimestamp variable, and several others.

Usual: We intended to use the `proxy` but added a better name to indicate the contact we want to use in commit bbaf4bf.

Cantina Managed: Fixed.

3.3.5 Add safety check for set `maxdepegthreshold` in `abstractoracle` contract

Severity: Low Risk

Context: AbstractOracle.sol#L116

Description: The `setMaxDepegThreshold()` function in the `AbstractOracle` contract does not include a safety check for the `maxDepegThreshold` to limit it to a maximum value of `10_000`. If set to a higher value, the `_checkDepegPrice()` function could revert due to an underflow in `SCALAR_ONE - threshold`.

Recommendation: It's recommended to ensure that `maxDepegThreshold` does not exceed `10_000`:

```
  function setMaxDepegThreshold(uint256 maxAuthorizedDepegPrice) external virtual {
      // ...
      if ($.maxDepegThreshold == maxAuthorizedDepegPrice) revert SameValue();
+     if ($.maxDepegThreshold > BASIS_POINT_BASE) revert InvalidMaxDepegThreshold();
  }
```

Usual: Fixed in commit bbaf4bf.

Cantina Managed: Fixed.

3.3.6 `cbrcoef` might be inaccurate if `treasury` sends/receives rwas outside of its usual swaps/redeems within `daocollateral`

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: When `CBRCoef` is calculated it fetches the Treasury's current balance for each of the valid RWAs it holds. The problem is that these values could change and be inaccurate if Treasury swaps/receives/sends any of the RWAs it holds, leading to an inaccurate `CBRCoef`

It would be ideal if the RWA balances were tracked internally instead (upon every swap/redeem), so even if Treasury sends/receives any RWA, it would not affect `DAOCollateral`'s accounting.

Recommendation: Consider tracking each of the RWA's balances internally. Also, consider adding an admin-only `adjustTreasury` function which will adjust any of the values if deemed necessary.

Usual: We removed the `cbrCoef` calculation from the contract. Therefore, this issue was fixed in commit 237f6cd.

Cantina Managed: Fixed.

3.3.7 Attacker can front-run calls to `selfpermit` and cause dos

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: `DaoCollateral` inherits both `Multicall` and `selfPermit` to allow users to give approval and swap/ redeem all in one transaction. The problem is that anyone is able to front-run the user's transaction and use the `permit` directly, causing the user's transaction to revert.

Recommendation: Put the `permit` call in a `try-catch` statement

Usual: Fixed in commit cddc9b0. A `swapWithPermit` function as been added together with a `try/catch` block. The `SelfPermit` contract will be removed.

Cantina Managed: Fixed.

3.4 Gas Optimization

3.4.1 `_getquoteinusd` could be implemented with one normalization less

Severity: Gas Optimization

Context: DaoCollateral.sol#L464

Description: This `_getQuoteInUsd` could be implemented in another way with one less normalization:

Recommendation:

```
function _getQuoteInUsd(uint256 tokenAmount, address rwaToken)
    internal
    view
    returns (uint256 wadAmountInUSD)
{
    (uint256 wadPriceInUSD, uint256 decimals) = _getPriceAndDecimals(rwaToken);
    return Math.mulDiv(tokenAmount, wadPriceInUSD, 10 ** decimals, Math.Rounding.Floor);
}
```

Usual: Acknowledged.

Cantina Managed: Resolved.

3.5 Informational

3.5.1 Permissioned launch restrictions and limitations

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: These are some considerations about the permissioned launch.

1. There are no secondary markets planned for Usd0 in the permissioned launch.
2. This restriction is possible because `Usd0` itself has an `allowlist`.
3. The `DAOCollateral` contract will not include any stablecoins, which means that the depeg check will never revert (see: `AbstractOracle._checkDepegPrice`). In case any stablecoin is added as collateral it could revert while calling `activateCBR()`.
4. The permissioned launch is primarily focused on `USYC` as collateral.
5. The oracle interface has been reviewed under the assumption of only Chainlink oracles.

Usual: Acknowledged.

Cantina Managed: Acknowledged.

3.5.2 Function `getquote()` might return incorrect precision

Severity: Informational

Context: AbstractOracle.sol#L132

Description: The function `getQuote()` in the `AbstractOracle` contract is only used in tests. However, it returns in the token's precision, not necessarily in 18 decimals. This could potentially break integrations to this contract.

Recommendation: It's recommended to either set the token precision to 18 decimals or remove this function.

Usual: Fixed in commit 18e1f7f.

Cantina Managed: Fixed.

3.5.3 Coding style: consistent return value pattern

Severity: Informational

Context: normalize.sol#L57

Description: Currently, both ways to return values in Solidity are used:

```
function option1() public returns (uint256 wadAmount) {
  //...
  wadAmount = /*...*/;
}

function option2() public returns (uint256) {
  return wadAmount;
}
```

Recommendation: For consistency, we recommend sticking to one pattern.

Usual: Fixed in commit 18e1f7f.

Cantina Managed: Fixed.

3.5.4 Duplicated code: `tokenamounttowad` and `wadamounttodecimals` could just call `tokenamounttodecimals`

Severity: Informational

Context: normalize.sol#L34

Description: There is no need to re-implement the decimal convert logic in `tokenAmountToWad` and `wadAmountToDecimals`.

Recommendation: Both functions could just use the existing `tokenAmountToDecimals` in `normalize.sol`

```
function tokenAmountToWad(uint256 tokenAmount, uint256 tokenDecimals)
    internal
    pure
    returns (uint256) {
    return tokenAmountToDecimals(tokenAmount, uint8(tokenDecimals), 18);
}

function wadAmountToDecimals(uint256 wadAmount, uint8 targetDecimals)
    internal
    pure
    returns (uint256) {
    return tokenAmountToDecimals(wadAmount, 18, targetDecimals);
}
```

Usual: Fixed in commit 18e1f7f.

Cantina Managed: Fixed.

3.5.5 Code consistency: `tokendecimals` could be `uint8` like in the other functions in `normalize.sol`

Severity: Informational

Context: normalize.sol#L34

Description: `uint8` and `uint256` types are used for token decimals.

Recommendation: The tokenDecimal parameter in `tokenAmountToWad` could be `uint8` like in other places in the codebase.

Usual: Fixed in commit 9e3f22a.

Cantina Managed: Fixed.

3.5.6 `notallowlisted()` error should be used instead of `notauthorized()`

Severity: Informational

Context: Usd0.sol#L172

Description: The `NotAllowlisted()` error is imported in `Usd0` contract but not used. Instead, the `NotAuthorized()` error is used in the `_update()` function when checking the `ALLOWLISTED` role. This should be replaced with the `NotAllowlisted()` error.

Recommendation: It's recommended to replace `NotAuthorized()` to `NotAllowlisted()`.

Usual: Fixed in commit 9e3f22a.

Cantina Managed: Fixed.

3.5.7 Skip address(0) in `_update()`

Severity: Informational

Context: Usd0.sol#L171-L176

Description: Before updating the balances, a validation checks whether both `from` and `to` addresses have the `ALLOWLISTED` role. In the tests, the deployment includes adding the `ALLOWLISTED` role to `address(0)`. However, if these lines were removed, the functionality of minting and burning would be broken. It's more efficient to directly fix this issue in the code by skipping the role verification when the `from` or `to` is `address(0)`.

Recommendation: It's recommended to skip the check for `ALLOWLISTED` when the `from` or `to` are `address(0)`:

```
- if (!$.registryAccess.hasRole(ALLOWLISTED, from)) {
+ if (!$.registryAccess.hasRole(ALLOWLISTED, from) && from != address(0)) {
      revert NotAuthorized();
  }
- if (!$.registryAccess.hasRole(ALLOWLISTED, to)) {
+ if (!$.registryAccess.hasRole(ALLOWLISTED, to) && to != address(0)) {
      revert NotAuthorized();
  }
```

Usual: Fixed in commit 9e3f22a.

Cantina Managed: Fixed.

3.5.8 Apply best practices in function flow

Severity: Informational

Context: ClassicalOracle.sol#L58

Description: As a best practice, it's recommended to start with access control, followed by parameter validation. The function `ClassicalOracle- initializeTokenOracle()` does not follow these best practices.

Recommendation: As a best practice, it's recommended to start with access control, followed by parameter validation.

Usual: Fixed in commit 9e3f22a.

Cantina Managed: Fixed.

3.5.9 Renaming variables to improve code understanding

Severity: Informational

Context: AbstractOracle.sol#L125

Description: Some variables can be easily confused due to their similar functions, such as `AbstractOracle.getPrice()` and `DAOCollateral._getPriceAndDecimals()`. To improve code readability, note that `_getPriceAndDecimals` has the same input parameter, `rwaToken`, and also returns a `price` and `decimals`. However, these `decimals` are the `rwaToken.decimals`. In `AbstractOracle.getPrice()`, `decimalsPrice` should be used instead of `decimals` as it relates to the price decimals.

Recommendation: It is recommended to rename the `decimals` to `decimalsPrice`:

```
  function getPrice(address token) public view override returns (uint256) {
-     (uint256 price, uint256 decimals) = _latestRoundData(token);
+     (uint256 price, uint256 decimalsPrice) = _latestRoundData(token);
      // ...
  }
```

Usual: Fixed in commit 9e3f22a.

Cantina Managed: Fixed.

3.5.10 Unused constants and errors

Severity: Informational

Context: constants.sol#L3, errors.sol#L3

Description: There are some declared constants and errors that should not be in the main codebase, as they are either unused or used only in scripts, mocks, or tests. Those used should be separated into a different file and placed in their respective folders.

Recommendation: It's recommended to remove or separate all unused code from the main codebase.

Usual: Fixed in commit bbaf4bf.

Cantina Managed: Fixed.

3.5.11 No way to remove an active `usd0rwa`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Currently, within `TokenMapping`, RWAs can only be added, but there's no way to remove any of them. In case the protocol has reached its limit of RWAs and wishes to change one of the used ones, there's no way to do so.

Recommendation: Consider adding a `removeUSD0Rwa` function.

Usual: Acknowledged. This design is intended.

Cantina Managed: Resolved.