Usual Pegasus

Security Review

Cantina Managed review by:

Xmxanuel, Lead Security Researcher
Akshay Srivastav, Security Researcher

March 11, 2025

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.

From Feb 16th to Feb 19th the Cantina team conducted a review of usual-pegasus(8fde95) on commit hash 8fde9572. The team identified a total of **27** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	2	1	1
Medium Risk	6	3	3
Low Risk	11	7	4
Gas Optimizations	0	0	0
Informational	8	3	5
Total	**27**	**14**	**13**

3 Findings

3.1 High Risk

3.1.1 Using `$.iUsd0ppVault.totalAssets()` in the `DistributionModule` is either incorrect or the vault is not following ERC4626

Severity: High Risk

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

Description: In the `ERC4626` standard the `totalAssets` should return the total amount of underlying assets held by the vault. In our understanding the `asset` in the context of the `iUsd0ppVault` will be the `USD0++` token. Since, functions like `vault.deposit` require the amount of USD0++ as parameter. In the deposit function the `USD0++` will be unwrap to `USD0` in the vault and then swapped to `sUSDe`. In the `_calculateVaultValueInUSD` function the `totalAssets` are multiplied with the `sUSDe` price.

However, this would be incorrect because the `sUSDe` price should be multiplied with the `sUSDe` token balance from the vault and not with the `USD0++` amount:

```
uint256 totalAssets = $.iUsd0ppVault.totalAssets();

// Check if the price is valid before calculating vaultValueUSD
if (iUsd0ppPrice > 0) {
  // Correctly scale the result to 18 decimals
  vaultValueUSD = (totalAssets * iUsd0ppPrice) / 1e18;
}
```

During the review, the concrete `iUsd0ppVault` was not available but we should consider a `ERC4626` vault. If the `totalAssets` would return the balance of the `sUSDe` tokens, it would not follow the ERC4626 standard but the calculation would be correct. Furthermore, it is not completely clear which concrete oracle will in the `DistributionModule` because the `iUsd0ppVault` address is used as a `key` in the `ClassicalOracle` contract in Usual and not the token itself.

```
$.oracle.getPrice(address($.iUsd0ppVault))
```

The `ClassicalOracle` has the actual Chainlink oracle address stored as `dataSource`. The `key` for the `getPrice` function should be the `sUSDe` address to be more exact.

```
$.oracle.getPrice(address(sUSDe))
```

Theoretically, it would be also possible to use a `USDe` (stablecoin) Chainlink price oracle.

1. Get the amount of `sUSDe` from the `iUsd0ppVault`.
2. Call the Ethena vault function `vault.convertToAssets` to get the `USDe` asset amount.
3. Multiple the `USDe` asset amount with the `USDe` price.

However, the current implementation logic requires the usage of the `sUSDe` oracle.

Recommendation: If the `iUsd0ppVault` should follow the ERC4626 standard the `vaultValueUSD` calculation should be moved to the `vault` itself. The `totalAssets` function would multiple the amount of `sUSDe` with the `price` and return the corresponding `USD0++` amount. The returned `vault.totalAssets()` value could be returned by the `_calculateVaultValueInUSD` since the protocols assume 1 dollar equals 1 Usd0++ instead of multiplying it with a price.

The other option would be to use another new function in the vault to return the `sUSDe` balance or by just calling `sUSDe.balanceOf(iUsd0ppVault)` instead of calling `vault.totalAssets`. The `ClassicalOracle` should use the `sUSDe` address as `token` in the `initializeTokenOracle` function.

Usual Labs: Acknowledged for now. We converted vault-related issues to tasks for the next sprints.

Cantina Managed: Acknowledged.

3.1.2 DoS of Usual distribution when `iUsd0ppVault` is removed

Severity: High Risk

Context: DistributionModule.sol#L958, DistributionModule.sol#L1351

Description: The `DistributionModule._calculateVaultValueInUSD` function calls the `iUsd0ppVault.totalAssets()` function. This call to vault contract will revert when `iUsd0ppVault` is `address(0)`. Few notes about the protocol:

- The acceptance criteria of `iUSD0++` distribution mentions that "an admin can set up/update/remove vault and oracle addresses".
- There exists a `updateIUsd0ppVault` function to change or remove `iUsd0ppVault`.
- The `distributeUsualToBuckets` function calls `_updateiUsd0ppDistributionShare` function which calls `_calculateiUSD0ppShare` function.

In case the admin removes the vault address from `DistributionModule` via `updateIUsd0ppVault` then the `iUsd0ppVault.totalAssets()` call will revert resulting in the revert of `distributeUsualToBuckets` call. Hence the Usual token distribution will be halted, i.e. a DoS situation.

Recommendation: Consider moving the zero address checks from `_calculateTotalSupply` to `_calculateVaultValueInUSD`.

```
  function _calculateTotalSupply(uint256 usd0ppSupply) internal view returns (uint256) {
      DistributionModuleStorageV0 storage $ = _distributionModuleStorageV0();

-     // If oracle or vault not set, return only USD0++ supply
-     if (address($.oracle) == address(0) || address($.iUsd0ppVault) == address(0)) {
-         return usd0ppSupply;
-     }

      return usd0ppSupply + _calculateVaultValueInUSD($);
  }

  function _calculateVaultValueInUSD(DistributionModuleStorageV0 storage $)
      internal
      view
      returns (uint256 vaultValueUSD)
  {
+     // If oracle or vault not set, return 0
+     if (address($.oracle) == address(0) || address($.iUsd0ppVault) == address(0)) {
+         return 0;
+     }
      // ...
  }
```

Usual Labs: Fixed in PR 2347.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 Treasuring can be added and removed when the `YieldModule` is paused

Severity: Medium Risk

Context: YieldModule.sol#L211, YieldModule.sol#L233

Description: The `addTreasury` and `removeTreasury` functions of `YieldModule` contract lack `whenNotPaused` modifier due to which the treasury accounts can be added and removed when the `YieldModule` contract is paused.

Scenario:

- The `YIELD_MODULE_SUPER_ADMIN_ROLE` account gets compromised.
- To prevent any damage the `PAUSING_CONTRACTS_ROLE` pauses the `YieldModule` contract.
- The `YIELD_MODULE_SUPER_ADMIN_ROLE` can still add or remove treasuries.

Recommendation: Consider adding `whenNotPaused` modifier to `addTreasury` and `removeTreasury` functions.

Usual Labs: Fixed in PR 2329.

Cantina Managed: Fix verified.

3.2.2 `YieldModule`: `YIELD_MODULE_TOKENOMICS_OPERATOR_ROLE` can switch the yield source of assets

Severity: Medium Risk

Context: YieldModule.sol#L196-L198, YieldModule.sol#L255-L267

Description: The intention of the `YieldModule` contract is to only allow the `YIELD_MODULE_SUPER_ADMIN_ROLE` to switch an oracle rate based source to a manual rate source. The `PR2151` description also mentions this as a spec:

> Overrides Yield Source interest when it is already via oracle - only super admin.

But the `YIELD_MODULE_TOKENOMICS_OPERATOR_ROLE` also has the ability to toggle a source. This can be done by first removing an existing source (via `removeYieldSource`) and then adding it again with different configuration (via `addYieldSourceWith**`).

Recommendation: Consider restricting the `removeYieldSource` function to `YIELD_MODULE_SUPER_ADMIN_ROLE`. If needed the `addYieldSourceWith**` and `updateFeed` functions can also be restricted to super admin.

Usual Labs: Fixed in PR 2368.

Cantina Managed: Fix verified.

3.2.3 Incorrect behaviour in `removeYieldSource` when removing non-last `yield source`

Severity: Medium Risk

Context: YieldModule.sol#L204-L205

Description: The `removeYieldSource` logic is incorrect if the removed asset is not the last element which has been added. There is a `idToYieldSource` mapping which uses the current `yieldSourceCount` as id to make the `yieldSources` iterable. If a `yieldSource` is removed which is not the last element the `getAllYieldSourceData` will return an empty entry and the last `yield source` will be missing.

Recommendation: The `idToYieldSource` mapping entry of the removed yield source needs to be overwritten with the last added entry in `idToYieldSource`. The following part needs to be added:

```
for (uint256 i = 0; i < $.yieldSourceCount;) {
    if ($.idToYieldSource[i] == asset) {
        // Move last yield source to current position, yield source order is not important
        $.idToYieldSource[i] = $.idToYieldSource[$.yieldSourceCount - 1];
        // Delete last position
        delete $.idToYieldSource[$.yieldSourceCount - 1];
        $.yieldSourceCount--;
        break;
    }
    unchecked {
        ++i;
    }
}
```

Usual Labs: Acknowledged, after storage simplification there is no need for this fix.

Cantina Managed: Acknowledged.

3.2.4 `sUSDe` can be used to `mint` USD0 or `vault target asset` is not considered in the blended weekly interest rate

Severity: Medium Risk

Context: YieldModule.sol#L326

Description: For an RWA asset to be considered in the blended weekly interest rate calculation, it must be added to the `tokenMappings` contract. This is true for RWAs as for the `iUsd0ppVault` underyling target asset (sUSDe).

The `getBlendedWeeklyInterest` function gets `RWAs` to iterate from the `tokenMapping` contract.

```
address[] memory rwas = $.tokenMapping.getAllUsd0Rwa();
```

However, adding new entry to the `tokenMapping` enables the token to be used as collateral to `mint` USD0. Additionally, a price oracle must exist to ensure the yield module functions correctly.

In the current implementation, this means that `sUSDe` could be used as collateral to mint `USD0`, which is not part of the indented vault design in Usual.

If the `sUSDe` is not added to the `tokenMapping` its interest rate will not be included in the blended interest rate calculation.

Recommendation: If the `sUSDe` should not be a new collateral type to mint USD0, the `tokenMapping` could include a new boolean flag, indicating whether an asset can be used as collateral for minting USD0.

If the `tokenMapping` is not an option, another approach would be to track the `RWAs` separately within the `yield module` contract.

Usual Labs: Acknowledged. We are aware of the current behavior, although we might change it in the future depending on the new vault version.

Cantina Managed: Acknowledged.

3.2.5 `DistributionModule._calculateVaultValueInUSD` doesn't consider swap fees in the `vault`

Severity: Medium Risk

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

Description: The `_calculateVaultValueInUSD` function in the `DistributionModule` should calculate the value of tokens held by the `iUsd0ppVault`. The `iUsd0ppVault` unwraps deposited USD0++ tokens into USD0. Afterwards the `USD0` is swapped to `USDe`, which is then staked to receive `sUSDe`. Since the redeem/withdraw function performs the reverse operation, `USD0++` is returned from the vault, users are eligible to receive `Usual` rewards for their `USD0++` deposit. Currently, `vaultValueUSD` is calculated by multiplying the vault's `sUSDe` balance by the `sUSDe` price.

```
vaultValueUSD = (totalAssets * iUsd0ppPrice) / 1e18;
```

Note: There is another issue in the report, that `totalAssets` here are not the `sUSDe` balance of the vault.

However, the `vaultValueUSD` does not equal the `total USD0++ amount` if all users would redeem at this point in time. If the third party protocol used to swap `USD0/USDe` has fees, these fees are not considered in the `_calculateVaultValueInUSD` calculation.

Recommendation: If users are only eligible for Usual rewards for the amount of `USD0++` they could receive back from the vault at this specific point in time, the `fees` should be considered. The `DistributionModule` could manage a `fee` percentage parameter with a specific role which is used in the `vaultValueUSD` calculation for a more exact dollar value estimation of the vault.

Usual Labs: Acknowledged, but out of scope for this audit since vault hasn't been implemented yet. Converted to ticket for internal development.

Cantina Managed: Acknowledged.

3.2.6 Flash loans can be used to manipulate `iUSD0ppDistributionShareOfLbt`

Severity: Medium Risk

Context: DistributionModule.sol#L1409-L1412, DistributionModule.sol#L1437-L1442

Description: The `DistributionModule` determines the `iUSD0ppDistributionShareOfLbt` by the ratio of total value of `iUsd0ppVault` to total available supply of USD0++.

$$\text{iUSD0ppShare} = \frac{\text{iUsd0ppVaultUsdValue}}{\text{totalSupplyUsd0pp} + \text{iUsd0ppVaultUsdValue}}$$

This way of determining `iUSD0ppDistributionShareOfLbt` is susceptible to flash loan attacks. An entity can inflate the `iUSD0ppDistributionShareOfLbt` by acquiring instant USD0++ tokens and depositing them in the `iUsd0ppVault` vault.

Scenario:

1. Attacker gets flash loan of USD0++ tokens (via Morpho and/or other protocols).
2. Deposits the USD0++ tokens into `iUsd0ppVault` vault.
3. Calls the `distributeUsualToBuckets` function. Since an inflated amount of USD0++ tokens is now deposited in vault, the `_calculateiUSD0ppShare` will return an inflated share value.
4. The inflated `iUSD0ppDistributionShareOfLbt` value is set in `DistributionModule`.
5. Attacker withdraws the USD0++ from vault and returns the flash loan.

After this incident, if the offchain Usual rewards are calculated using the inflated `iUSD0ppDistributionShareOfLbt` value then an inflated reward share will be given to iUSD0++ holders. If the inflated value is discarded then the legitimate rewards of iUSD0++ holders will also be forfeited.

Recommendation: Consider restricting the `distributeUsualToBuckets` call to an operator role. Or, since the `iUSD0ppDistributionShareOfLbt` value is not used at smart contract level consider removing it completely. The share value can be computed off-chain by querying past token supplies.

Usual Labs: We don't see a risk here because a flash loan will never be profitable and we can fix the issue at any time.

Cantina Managed: Acknowledged.

3.3 Low Risk

3.3.1 `YieldModule` can return blended interest rate and p90 interest rate in a paused state

Severity: Low Risk

Context: YieldModule.sol#L321, YieldModule.sol#L353

Description: The `getBlendedWeeklyInterest` and `getP90InterestRate` functions of `YieldModule` lacks `whenNotPaused` modifier due to which the contract can return interest rates even in the paused state. These interest values are consumed by `DistributionModule` to determine Usual token distribution. A contract is paused in critical scenarios (bug disclosures, active exploits, etc). Returning crucial data in paused state that is consumed by other protocol components can lead to unintended outcomes.

Recommendation: Consider adding `whenNotPaused` modifier to `getBlendedWeeklyInterest` and `getP90InterestRate` functions.

Usual Labs: Valid: If protocol is in paused state we should revert when fetching p90 IR or blended interest. Fixed in PR 2331.

Cantina Managed: Fix verified.

3.3.2 `YieldModule`: no cap on interest rates returned by external oracle

Severity: Low Risk

Context: YieldModule.sol#L182-L188, YieldModule.sol#L255-L258, YieldModule.sol#L348-L349, YieldModule.sol#L493

Description: In the `addYieldSourceWithWeeklyInterest` and `updateInterestRate` functions of `YieldModule` it is enforced that the input `weeklyInterestBps` does not exceed `BASIS_POINT_BASE` (100%), but no such bound is applied on the interest rate returned by an external oracle. The oracle can return any large value as the weekly interest rate for a yield source. This will significantly impact the blended weekly interest rate returned by `YieldModule` contract.

Recommendation:

1. Consider capping the oracle returned interest rate to an acceptable maximum interest rate value.

```
if (oracleRate > BASIS_POINT_BASE) {
    revert InvalidWeeklyInterestBps();
}
```

2. Further an invariant can also be added in `getBlendedWeeklyInterest` to ensure that the blended rate does not exceed an accepted maximum interest rate threshold.

```
function getBlendedWeeklyInterest() public view override returns (uint256 blendedRate) {
    // ...
    blendedRate = totalValue > 0 ? Math.mulDiv(weightedSum, 1, totalValue, Math.Rounding.Floor) : 0;

    assert(blendedRate != 0 && blendedRate <= BASIS_POINT_BASE);
}
```

Usual Labs: Fixed in PR 2336.

Cantina Managed: Fix verified.

3.3.3 Missing resetting the manual rate when manual rate source is switched to oracle rate

Severity: Low Risk

Context: YieldModule.sol#L281-L290

Description: The `YieldModule.updateFeed` function can be used to switch a manual rate source to a oracle based yield source. When that is done the old `YieldSourceData.weeklyInterestBps` state value is not reset, leading to an outdated interest rate value being stored in contract storage for a yield source.

Recommendation: Consider resetting the `YieldSourceData.weeklyInterestBps` state in `updateFeed` function.

```
  function updateFeed(address asset, address feedAddress) external whenNotPaused {
      YieldModuleStorageV0 storage $ = _yieldModuleStorageV0();
      $.registryAccess.onlyMatchingRole(YIELD_MODULE_TOKENOMICS_OPERATOR_ROLE);

      YieldSourceData storage source = _getYieldSource(asset);
      _validateFeedInterface(feedAddress);

      source.feed = IAggregator(feedAddress);
      emit FeedUpdated(asset, feedAddress);

+     if (source.weeklyInterestBps != 0) {
+         source.weeklyInterestBps = 0;
+     }
  }
```

Moreover the `source.lastUpdated` may also be set to `block.timestamp` to store the last timestamp when the source was changed.

Usual Labs: Valid: Reset `weeklyInterestBps` when updating feed manually. Fixed in PR 2339.

Cantina Managed: Fix verified.

3.3.4 `YieldModule: YIELD_MODULE_TOKENOMICS_OPERATOR_ROLE` actions should be divided among multiple roles

Severity: Low Risk

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

Description: Currently the `YIELD_MODULE_TOKENOMICS_OPERATOR_ROLE` can perform these actions on the `YieldModule`:

- Add/remove/update yield sources.
- Set `MaxDataAge`.
- Update manual rates.

Most of these actions will be done once in a while except updating the manual rates. Updating of manual rates will be done very frequently (maybe daily/weekly/biweekly). It is likely that a multisig or an eoa will be granted this role which will post the manual rates in a fully/partially automated way. Having the ability to update yield sources on the same eoa/multisig seems risky.

Recommendation: Consider further separating the operator role into:

- One role to add/remove/update yield sources and set `MaxDataAge`. This role can be given to a governance/timelock like entity.

- Another role to post the manual interest rates onchain. This can be granted to an eoa/multisig.

Usual Labs: Fixed in PR 2368.

Cantina Managed: Fix verified.

3.3.5 Max interest cap inconsistency between yield and distribution module

Severity: Low Risk

Context: DistributionModule.sol#L765-L772, YieldModule.sol#L182-L188, YieldModule.sol#L255-L258

Description: The `YieldModule.updateInterestRate` accepts an interest rate of 100% but the `DistributionModule.distributeUsualToBuckets` only works when the interest rate is less than 100%. In case only one RWA asset exists and its interest rate is set to `100%` then usual distribution transactions will start reverting.

Recommendation: Consider implementing a consistent max interest rate cap in yield and distribution modules. Either both should accept the `100%` value or both should reject it.

Usual Labs: Fixed in PR 2340.

Cantina Managed: Fix verified.

3.3.6 Missing `yieldSource` will revert `getBlendedWeeklyInterest`

Severity: Low Risk

Context: YieldModule.sol#L489

Description: In the current design, if a new `RWA` gets added to the `tokenMapping` but no corresponding `yieldSource` gets added to the `YieldModule` the `getBlendedWeeklyInterest` would revert with `YieldSourceDataTooOld`.

Since, the existing `_getYieldSource` function, which checks wether a `yieldSource` exists, is not used in '_getYieldSourceRateAndAmount, we assume this is intentional to prevent reversion in such cases.

The `DEFAULT_YIELD_FEED_RATE = 0;` rate of zero could be used in such a case. However, currently the `_getYieldSourceRateAndAmount` would revert later with `YieldSourceDataTooOld`.

Recommendation: Use the `DEFAULT_YIELD_FEED_RATE` for missing `yieldSources` or alternatively if the function should revert use the existing `_getYieldSource` function in `_getYieldSourceRateAndAmount` which checks if a yield source exists. This would ensure the function throws the correct corresponding error, `YieldSourceNotFound`.

Usual Labs: Fixed in PR 2350.

Cantina Managed: Fix verified.

3.3.7 Unbounded `p90Rate` can be set leading to distribution failure

Severity: Low Risk

Context: DistributionModule.sol#L774-L776, YieldModule.sol#L305-L313

Description: `YieldModule.setP90InterestRate` function allows any `uint256` value to be set as `p90Rate`. It can technically be anything from `0` to `uint256.max`. Also note that the `DistributionModule.distributeUsualToBuckets` function reverts if `(p90Rate == 0 || p90Rate >= BPS_SCALAR)`.

In case an invalid value (`0` or `>= BPS_SCALAR`) is set as the `p90Rate` in `YieldModule` then `DistributionModule.distributeUsualToBuckets` calls will start reverting leading to a halted Usual token distribution state.

Recommendation:

```
  function setP90InterestRate(uint256 p90Rate) external override whenNotPaused {
      YieldModuleStorageV0 storage $ = _yieldModuleStorageV0();
      $.registryAccess.onlyMatchingRole(YIELD_MODULE_P90_INTEREST_ROLE);
+     if (p90Rate == 0 || p90Rate >= BASIS_POINT_BASE) {
+         revert InvalidInput();
+     }
      if ($.p90Rate == p90Rate) revert SameValue();

      $.p90Rate = p90Rate;

      emit P90InterestRateSet(p90Rate);
  }
```

Usual Labs: Fixed in PR 2332.

Cantina Managed: Fix verified.

3.3.8 `vaultValueUSD` calculation in `DistributionModule` assume a `1e18` precision of the token

Severity: Low Risk

Context: DistributionModule.sol#L1356

Description: The `vaultValueUSD = (totalAssets * iUsd0ppPrice) / 1e18;` assume a precision of 18 decimals for the `totalAsset`. The price will be always 18 decimals. However, if a vault is integrated with another token like USDC (6 decimals) precision the calculation would be incorrect.

Recommendation: If the integration assumes that it could be any ERC4646 vault. The calculation should work with any token precision. The correct calculation would be:

```
vaultValueUSD = (totalAssets * iUsd0ppPrice) / 10 ** IERC20Metadata(targetAssetToken).decimals();
```

Usual Labs: Acknowledged. In the current vault version the `asset` function returns the `sUSDe` address. This finding will be considered in the refactoring of the vault.

Cantina Managed: Acknowledged.

3.3.9 `ClassicalOracle.getPrice` will revert in case of a stablecoin depeg in `getBlendedWeeklyInterest`

Severity: Low Risk

Context: YieldModule.sol#L334

Description: The `YieldModule` uses the `ClassicalOracle` contract to communicate with the actual oracle to retrieve the `price` for an `RWA`.

```
uint256 price = $.oracle.getPrice(rwa);
```

The `ClassicalOracle.getPrice` implementation will revert if the `RWA` is a stablecoin that has depegged:

```
function getPrice(address token) public view override returns (uint256) {
    (uint256 price, uint256 decimalsPrice) = _latestRoundData(token);
    price = price.tokenAmountToWad(uint8(decimalsPrice));
    _checkDepegPrice(token, price);
    return price;
}
```

The `_checkDepegPrice` function reverts if the `oracle` is for a stablecoin and the stablecoin has depegged. This revert make sense in the context of, for example a `DAOCollateral.redeem` call where users should not be able to redeem to a depegged stablecoin. However, in the context of calculating the blended interest rate, the calculation could accept the lower value for the a depegged stablecoin and continue the calculation.

Recommendation: In case a stablecoin gets added as collateral type to Usual consider the usage of a separate `oracle.getPrice` function which would not revert in case of a depeg event.

Usual Labs: Acknowledged. In the current design, we want to revert the `Usual distribution` as well.

Cantina Managed: Acknowledged.

3.3.10 Missing swing check for vault oracle price

Severity: Low Risk

Context: DistributionModule.sol#L1338-L1361

Description: The `iUSD0++ spec` mentions that the protocol should be able to handle oracle edge cases including large price swings.

> Demonstrate that oracle edge cases can be handled, i.e. what happens in cases of:
> 1. Staleness
> 2. Swings

However no such price swing check is present in the `DistributionModule` for vault oracle price.

Usual Labs: Acknowledged. Our Tokenomics team said it's alright as it is, we have CBR and p90. They might revisit the spec once again in the future.

Cantina Managed: Acknowledged.

3.3.11 `updateIUsd0ppVault` doesn't update the `iUSD0ppDistributionShareOfLbt` value

Severity: Low Risk

Context: DistributionModule.sol#L958

Description: If the `vault` gets updated it most likely means the stored `iUSD0ppDistributionShareOfLbt` will be incorrect/outdated as well. Since the off-chain calculation will use the `iUSD0ppDistributionShareOfLbt` this can result in errors.

Recommendation: The update function could be called as part of `updateIUsd0ppVault` to correctly update the `iUsd0ppDistributionShare` as well.

```
_updateiUsd0ppDistributionShare($);
```

If the `iUsd0pDistributionShare` is more like a snapshot from the last distribution call, then it would be fine not to update it. Since, the last `Usual` distribution should be based on the old vault `iUsd0ppDistributionShare` value.

Usual Labs: Acknowledged. It will be considered together with the implementation of the new `vault`.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Non-existent `treasury` can be passed to `YieldModule.removeTreasury` function

Severity: Informational

Context: YieldModule.sol#L211-L230

Description: The `YieldModule.removeTreasury` function does not validate the existence of the input `treasury` address in the treasury list. If a non-existent `treasury` address is provided as input then the `removeTreasury` gets executed successfully as a `no-op`.

Recommendation: Consider reverting if the `treasury` address is invalid.

Usual Labs: Fixed in PR 2330.

Cantina Managed: Fix verified.

3.4.2 Storage struct of `YieldModule` can be simplified

Severity: Informational

Context: YieldModule.sol#L90-L113

Description: The current storage struct of `YieldModule` contract is overcomplex. There are multiple state variables which can be removed or simplified, as:

- `idToYieldSource` is never really used and isn't needed for contract's functioning.
- `yieldSourceCount` is not needed as you always iterate over the `rwas` returned by `tokenMapping`.
- `idToTreasury` and `treasuryCount` can be replaced with an address array to achieve the same results.
- `registryContract` is never used after initialization.
- `YieldSourceData.asset` is never used.

The current storage struct can be replaced with a simpler struct like:

```
struct YieldModuleStorageV0 {
    IRegistryAccess registryAccess;
    IOracle oracle;
    ITokenMapping tokenMapping;
    uint256 maxDataAge;
    uint256 p90Rate;
    mapping(address => YieldSourceData) yieldSources;
    address[] treasuries;
}
struct YieldSourceData {
    uint256 lastUpdated;
    uint256 weeklyInterestBps;
    IAggregator feed;
}
```

Usual Labs: Storage simplification done in PR 2346.

Cantina Managed: Fix verified.

3.4.3 `YieldModule`: `addYieldSourceWithWeeklyInterest` can be called with `0` as the `weeklyInterestBps`

Severity: Informational

Context: YieldModule.sol#L182-L193

Description: The `YieldModule.addYieldSourceWithWeeklyInterest` allows the admin to pass `0` as the `weeklyInterestBps` for an `asset`. Using this a new yield source can be added with `address(0)` as the `feed` and `0` as the manual interest rate.

Recommendation: This could be an intended case to prepare `YieldModule` for an upcoming RWA. If that's not the case then do revert when `weeklyInterestBps` is `0`.

Usual Labs: Acknowledged. A `weeklyInterestBps` of zero should be possible.

Cantina Managed: Acknowledged.

3.4.4 `YieldModule`: manual weekly yield of assets is capped at `100%`

Severity: Informational

Context: YieldModule.sol#L182-L188, YieldModule.sol#L255-L258

Description: In `addYieldSourceWithFeed` and `updateInterestRate` functions of `YieldModule`, it is enforced that the input interest rate does not exceed `BASIS_POINT_BASE` (100%). Technically the interest offered by any asset can be significantly more than `100%`. Practically for RWAs it is unlikely that their weekly interest rate will exceed the threshold. But in case any asset offers an interest rate of more than 100% then YieldModule won't be able to support that.

Recommendation: Consider revisiting the interest rate cap for assets and raise it if necessary.

Usual Labs: Acknowledged. It's okay because no assets with a yield of over 100% will be added in the foreseeable future.

Cantina Managed: Acknowledged.

3.4.5 `YieldModule`: negative interest rate reported by oracle is ignored silently

Severity: Informational

Context: YieldModule.sol#L497

Description: In `YieldModule._getYieldSourceRateAndAmount` if the interest rate returned by oracle is negative then it is silently considered as `0`. For a manual rate asset there is no option to report a negative interest rate. Ideally in the case when one of the RWAs is generating negative yield (losses) then that should also be accounted in the net blended rate returned by yield module.

Recommendation: Consider emitting an event when oracle reports a negative rate. Moreover the module logic can be changed to also consider negative interest rate on blended rate calculations.

Usual Labs: Acknowledged. It doesn't require a change.

Cantina Managed: Acknowledged.

3.4.6 Non-critical issues & code improvements

Severity: Informational

Context: YieldModule.sol#L271, YieldModule.sol#L450

List of issues:

1. `YieldModule.sol#L271`: Misleading comment. Feed is not set here.
2. `YieldModule.sol#L450`: There is no benefit of using `try/catch` as call is simply reverted when external call fails. The revert error returned by the feed is also lost.
3. `DistributionModule.sol#L771`: Improve error types. `InvalidInput` is used for `ratet` and `p90Rate`.

Usual Labs: Fixed in PR 2366.

Cantina Managed: Fix verified.

3.4.7 `_calculateVaultValueInUSD` implementation is unnecessary complicated

Severity: Informational

Context: DistributionModule.sol#L1338

Description: The `_calculateVaultValueInUSD` function is unnecessarily complicated.

Note: This issue ignores other findings related to the correct calculation of the `_calculateVaultValueInUSD`.

Recommendation: Consider a simpler implementation to avoid unnecessary if conditions and local variables.

```
function _calculateVaultValueInUSD(DistributionModuleStorageV0 storage $)
    internal
    view
    returns (uint256 vaultValueUSD)
{
    try $.oracle.getPrice(address($.iUsd0ppVault)) returns (uint256 iUsd0ppPrice) {
        return ($.iUsd0ppVault.totalAssets() * iUsd0ppPrice) / 1e18;
    } catch {
        // If the oracle call reverts a zero price results in a zero vault value
        return 0;
    }
}
```

Usual Labs: Acknowledged. It will be considered together with the new `vault` implementation.

Cantina Managed: Acknowledged.

3.4.8 Consider a function in the `YieldModule` which returns both the `blendedWeeklyInterest` and the `p90Rate`

Severity: Informational

Context: DistributionModule.sol#L979

Description: Currently, the `blendedWeeklyInterest` and `p90Rate` is primarily used by the `DistributionModule`. The `DistributionModule` performs two calls to the yield module to get the values:

```
function _getRateAndP90Rate(DistributionModuleStorageV0 storage $)
    internal
    view
    returns (uint256, uint256)
{
    return ($.yieldModule.getBlendedWeeklyInterest(), $.yieldModule.getP90InterestRate());
}
```

Recommendation: Consider an additional getter function that returns the `blendedWeeklyInterest` and `p90Rate`.

Usual Labs: Acknowledged.

Cantina Managed: Acknowledged.