CANTINA

Usual Pegasus

Security Review

Cantina Managed review by:

Xmxanuel, Lead Security Researcher
Akshay Srivastav, Security Researcher

February 1, 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 Jan 24th to Jan 28th the Cantina team conducted a review of usual-pegasus on commit hash bb36f956. The team identified a total of **17** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	0	1
Medium Risk	1	1	0
Low Risk	11	8	3
Gas Optimizations	0	0	0
Informational	4	3	1
Total	**17**	**12**	**5**

3 Findings

3.1 High Risk

3.1.1 Locking of Usual tokens of integrators by bypassing redirection flow

Severity: High Risk

Context: DistributionModule.sol#L437

Description: The `DistributionModule.claimOffChainDistribution` is a publicly accessible function by which anyone can initiate the Usual token distribution for any eligible account.

The redirection logic has been implemented to redirect the Usual rewards of integrators to a different address, in case the integrator's native address is unable to handle Usual rewards.

Anyone can monitor the `initiatedRedirectedOffChainDistribution` & `initiatedRedirectedOffChainDistributionStartingTimestamp` states for accounts and call the `claimOffChainDistribution` function just before the redirection can be activated for an integrator, making the entire redirection logic ineffective. Due to this the Usual tokens will get minted to the integrator's account instead of its redirected recipient, leading to locking and loss of Usual tokens for the integrator.

Currently there are `~69M` Usual tokens eligible and waiting to be claimed as offchain distribution rewards. All Usual tokens of integrators that are planned to be redirected are susceptible to this attack.

Recommendation: One of the ways to resolve this is to disable the public access of `claimOffChainDistribution` function for an account when a redirection is initiated for that account. Once the redirection is activated then the public accessibility of `claimOffChainDistribution` function for that account can be re-enabled.

```
function claimOffChainDistribution(address account, ...) {
    if ($.initiatedRedirectedOffChainDistribution[account] != address(0)) revert();
    // ...
}
```

Please note that anyone can still front run the `redirectOffChainDistribution` call (which sets the `initiatedRedirectedOffChainDistribution` state) and initiate the claim for integrators.

Usual: Acknowledged.

Cantina Managed: Acknowledged.

3.2 Medium Risk

3.2.1 Active redirection can be removed when the `DistributionModule` contract is paused

Severity: Medium Risk

Context: DistributionModule.sol#L863-L871

Description: The `DistributionModule.removeRedirectedOffChainDistribution` function is missing `whenNotPaused` modifier. Due to this an active redirection can be removed by an account when the `DistributionModule` contract is paused.

Pausing of contract is only intended to be triggered in critical scenarios (exploit, upgrades, active bug, etc...) and performing any user operation in paused state can lead to unintended outcomes.

One scenario could be when the redirection admin address gets compromised. Pausing the contract would not prevent the compromised redirection admin from removing the redirections.

Recommendation:

```
- function removeRedirectedOffChainDistribution(address account) external {
+ function removeRedirectedOffChainDistribution(address account) external whenNotPaused {
      DistributionModuleStorageV0 storage $ = _distributionModuleStorageV0();
      if (msg.sender != account) {
          _requireOnlyRedirectionAdmin($);
      }

      delete $.activeRedirectedOffChainDistribution[account];
      emit OffChainDistributionRedirectedRemoved(account);
  }
```

Usual: Fixed in PR 2221.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 Frontrunning `initializeV1` function of `DistributionModule`

Severity: Low Risk

Context: DistributionModule.sol#L294

Description: The `DistributionModule.initializeV1` function is a publicly accessible function which sets the fee rates for the contract. This function is intended to be called just after the `DistributionModule` contract is upgraded to the new implementation.

It is possible for a malicious user to frontrun the `initializeV1` call and set the initial fee rates which will lead to incorrect fee distribution. The admins will need to call `setFeeRates` function to reset the fee rates to correct values.

Recommendation: Either.

- Perform the `initializeV1` call with the contract upgrade txn, i.e. in the `upgradeToAndCall` call.
- Add access restriction to the `initializeV1` function.

Usual: Not acknowledged, we use `upgradeToAndCall`.

Cantina Managed: Acknowledged.

3.3.2 Using `redirectOffChainDistribution` function to remove a redirection

Severity: Low Risk

Context: DistributionModule.sol#L822-L839

Description: The `DistributionModule.redirectOffChainDistribution` function can be used by admin to remove an existing active redirection by providing `address(0)` as the `newAccount`. By doing this the admin will be able to remove the redirection with a delay, moreover the account can also cancel the redirection removal if needed.

Since there is a dedicated function `removeRedirectedOffChainDistribution` for removing an existing redirection, the same should not be allowed in `redirectOffChainDistribution`.

Recommendation: Consider reverting if `newAccount` is `address(0)` in the `redirectOffChainDistribution` function.

Usual: Fixed in PR 2220.

Cantina Managed: Fix verified.

3.3.3 Incorrect value emitted in `OffChainDistributionRedirectedAccepted` event

Severity: Low Risk

Context: DistributionModule.sol#L894-L902

Description: The `DistributionModule.acceptRedirectedOffChainDistribution` function resets the `initiatedRedirectedOffChainDistribution` value to `0` and then uses it as a parameter for emitting

`OffChainDistributionRedirectedAccepted` event. Due to this a `0` value is always emitted by the function call.

Recommendation:

```
  emit OffChainDistributionRedirectedAccepted(
-     account, $.initiatedRedirectedOffChainDistribution[account]
+     account, $.activeRedirectedOffChainDistribution[account]
  );
```

Usual: Fixed in PR 2222.

Cantina Managed: Fix verified.

3.3.4 The `removeRedirectedOffChainDistribution` function can be called repeatedly

Severity: Low Risk

Context: DistributionModule.sol#L863-L871

Description: The `DistributionModule.removeRedirectedOffChainDistribution` does not check if a non-zero `activeRedirectedOffChainDistribution` state exist for an account before deleting that state. Due to this the `removeRedirectedOffChainDistribution` function can be called repeatedly by accounts who has `address(0)` as their `activeRedirectedOffChainDistribution`.

Recommendation:

```
  function removeRedirectedOffChainDistribution(address account) external {
      DistributionModuleStorageV0 storage $ = _distributionModuleStorageV0();
      if (msg.sender != account) {
          _requireOnlyRedirectionAdmin($);
      }
+     if ($.activeRedirectedOffChainDistribution[account] == address(0)) {
+         revert NoActiveRedirectedOffChainDistribution();
+     }

      delete $.activeRedirectedOffChainDistribution[account];
      emit OffChainDistributionRedirectedRemoved(account);
  }
```


Usual: Fixed in PR 2223.

Cantina Managed: Fix verified.

3.3.5 The `redirectOffChainDistribution` function can be called with same initiated redirection value again

Severity: Low Risk

Context: DistributionModule.sol#L822-L839

Description: For an account whose rewards redirection is already initiated (i.e. `initiatedRedirectedOffChainDistribution` state is set), the admin call the `redirectOffChainDistribution` function again with the same existing value. This will restart the redirection challenge period for the account causing additional delay for redirection acceptance. In case the admin wants to reinitiate an already initiated redirection then they should first cancel the initiated redirection and then initiate another redirection.

Recommendation:

```
  function redirectOffChainDistribution(address account, address newAccount)
      external
      whenNotPaused
  {
      DistributionModuleStorageV0 storage $ = _distributionModuleStorageV0();
      _requireOnlyRedirectionAdmin($);

+     if ($.initiatedRedirectedOffChainDistribution[account] != address(0) ||
↪  $.initiatedRedirectedOffChainDistributionStartingTimestamp[account] != 0) {
+         revert();
+     }

      if (newAccount == $.activeRedirectedOffChainDistribution[account]) {
          revert SameValue();
      }
      // ...
  }
```

Usual: Fixed in PR 2224.

Cantina Managed: Fix verified.

3.3.6 A new redirection can be initiated for an account with an existing active redirection

Severity: Low Risk

Context: DistributionModule.sol#L822-L839

Description: The `DistributionModule.redirectOffChainDistribution` function currently allows initiating a new redirection for an account with an existing active redirection. If an account already has an active redirection to recipient `R1` then admin can initiate a new redirection to a new recipient `R2`.

Also the natspec for the `redirectOffChainDistribution` functions says that "If the account is already redirected, the function will revert".

Since a dedicated `removeRedirectedOffChainDistribution` function is present, admin should use that to remove an active redirection and then initiate a new redirection.

Recommendation:

```
  function redirectOffChainDistribution(address account, address newAccount)
      external
      whenNotPaused
  {
      DistributionModuleStorageV0 storage $ = _distributionModuleStorageV0();
      _requireOnlyRedirectionAdmin($);

      if (newAccount == $.activeRedirectedOffChainDistribution[account]) {
          revert SameValue();
      }
      if (account == newAccount) {
          revert SameValue();
      }
+     if ($.activeRedirectedOffChainDistribution[account] != address(0)) {
+         revert AlreadyRedirected();
+     }

      $.initiatedRedirectedOffChainDistribution[account] = newAccount;
      $.initiatedRedirectedOffChainDistributionStartingTimestamp[account] = block.timestamp;
      emit OffChainDistributionRedirectInitialized(account, newAccount, block.timestamp);
  }
```

Usual: Fixed in PR 2224.

Cantina Managed: See this the PR 2246 comment.

3.3.7 The `newAccount` for the `cancelInitiatedRedirectedOffChainDistribution` event can be any value

Severity: Low Risk

Context: DistributionModule.sol#L842

Description: The `newAccount` parameter in `cancelInitiatedRedirectedOffChainDistribution` is only used for emitting the event. The newAccount address could also be incorrect since no validation happens.

Recommendation: A cleaner approach would be to remove the parameter and use the `initiatedRedirectedOffChainDistribution` mapping where the `newAccount` is stored.

```
function cancelInitiatedRedirectedOffChainDistribution(address account) {
  // ...
  emit OffChainDistributionRedirectedCancelled(account, $.initiatedRedirectedOffChainDistribution[account]);
  delete $.initiatedRedirectedOffChainDistribution[account];
  delete $.initiatedRedirectedOffChainDistributionStartingTimestamp[account];
}
```

Usual: Fixed in PR 2229.

Cantina Managed: Fix verified.

3.3.8 `DistributionModule:` missing allowances for redirect target address in redirect `cancel` and `remove`

Severity: Low Risk

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

Description: The redirect address feature allows the `admin` to redirect the Usual token payout to any new target address for a specific account. This feature has been built explicitly for contracts eligible to receive the tokens but can't process received tokens. If a contract cannot handle received `ERC20` tokens, it is highly likely that it won't be able to make a call to the `DistributionModule` either.

Currently, two functions allow the `account` (contract unable to handle Usual tokens) to call the `DistributionModule`:

- `cancelInitiatedRedirectedOffChainDistribution`.
- `removeRedirectedOffChainDistribution`.

These functions can otherwise only be called by the `RedirectionAdmin`.

Recommendation: As explained, there is a very low likelihood that the contracts will be able to perform this action themselves. If the intention behind this allowance is to enable the entity behind the contract (such as the governance of a DeFi protocol) to call `cancel` or `remove` themselves, it would make more sense to allow the redirected address stored in `activeRedirectedOffChainDistribution[account]` to call the function. Because the redirect target address will be controlled by the relevant entity behind the contract.

Usual: Acknowledged. It is working as intended.

Cantina Managed: Acknowledged.

3.3.9 Distributing the fees to `burn`, `yieldTreasury` and `UsualX` will result in Usual `DUST` in the `DistributionModule`

Severity: Low Risk

Context: DistributionModule.sol#L742

Description: The `DistributionModule` distributes collected Usual fees to the `yieldTreasury`, `UsualX` and burns a certain percentage. The three percentages individual are defined as basis points and are required to add up to `10_000` when setting them. When calculating the exact amount based on the individual basis points the amount is always uses `Math.Rounding.Floor` to round in favor of the protocol.

```
uint256 amount = Math.mulDiv(
    usualDistribution, $.usualXDistributionShare, BPS_SCALAR, Math.Rounding.Floor
);
```

However, this can still result in dust in the DistributionModule when the total fees (`feeSwept`) amount is not a multiple of `10_000`. Small amounts of `dust` can break protocol invariants or lead to further problems and should be avoided in the design if possible.

Proof of Concept: A test case to illustrate the `dust` and the fee distribution:

```
// test case will fail
function testFeePercentage() public {
    uint256 treasuryFeeRate = 3334;
    uint256 usualXFeeRate = 3333;
    uint256 burnFeeRate = 3333;

    uint256 feeSwept = 1_111_111_111_111_111_111_111;

    assertEq(treasuryFeeRate + usualXFeeRate + burnFeeRate, BASIS_POINT_BASE);

    uint256 feeAmountXFeeRate =
        Math.mulDiv(feeSwept, usualXFeeRate, BASIS_POINT_BASE, Math.Rounding.Floor);

    uint256 feeAmountTreasuryFeeRate =
        Math.mulDiv(feeSwept, treasuryFeeRate, BASIS_POINT_BASE, Math.Rounding.Floor);
    uint256 feeAmountBurnFeeRate =
        Math.mulDiv(feeSwept, burnFeeRate, BASIS_POINT_BASE, Math.Rounding.Floor);

    assertEq(feeAmountXFeeRate + feeAmountTreasuryFeeRate + feeAmountBurnFeeRate, feeSwept);
}
```

Recommendation: The dust in the fee distribution can be easily avoided by not defining a percentage for the `feeAmountBurnFeeRate`. The remaining amount after the `yieldTreasury` and `UsualX` fee should be burned. The `burnFeeAmount` will be defined implicitly after `feeAmountTreasuryFeeRate` and `feeAmountBurnFeeRate` are calculated as:

```
feeAmountBurnFeeRate = feeSwept - feeAmountTreasuryFeeRate -treasuryFeeRate;`
```

Usual: Fixed in PR 2233.

Cantina Managed: Fix verified.

3.3.10 Missing `FeeSwept` event in `Usd0PP.sweepFees`

Severity: Low Risk

Context: Usd0PP.sol#L655

Description: `Usd0PP.sweepFees` is missing a `FeeSwept` event like in `UsualX.sweepFees`.

Recommendation: Since both functions transfer fees to the `DistributionModule`. `USD0PP.sweepFees` should use the same event:

```
event FeeSwept(address indexed caller, address indexed collector, uint256 amount);
```

Usual: Fixed in PR 2235.

Cantina Managed: Fix verified.

3.3.11 The daily Usual distribution will fail if yield treasury is set to `address(0)`

Severity: Low Risk

Context: DistributionModule.sol#L770, DistributionModule.sol#L1022-L1030, RegistryContract.sol#L106-L112

Description: The Usual distribution which happens daily also tries to sweep fees from UsualX vault and send a portion of Usual tokens to yield treasury address.

In case the yield treasury address on `RegistryContract` is set to `address(0)` then the `registryContract.getContract(CONTRACT_YIELD_TREASURY)` call will revert, leading to reverting of the `distributeUsualToBuckets` call. Hence daily distribution of Usual tokens cannot be processed anymore.

The current onchain implementation of UsualX contract had these statements:

```
function sweepFees() external nonReentrant {
    UsualXStorageV0 storage $ = _usualXStorageV0();
    address yieldTreasury = $.registryContract.getContract(CONTRACT_YIELD_TREASURY);

    if (yieldTreasury == address(0)) {
        revert NullAddress();
    }
    // ...
}
```

which indicates that there might be a possibility that yield treasury address on `RegistryContract` can be `address(0)`.

Recommendation: Consider having the ability to process Usual token distribution even if the yield treasury is set to `address(0)`.

Usual: Acknowledged.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Missing getter function for `initiatedRedirectedOffChainDistribution` storage state

Severity: Informational

Context: DistributionModule.sol#L242-L243

Description: The `DistributionModule` contract is missing a getter function to read the `initiatedRedirectedOffChainDistribution` storage state.

Recommendation:

```
function getInitiatedRedirectedOffChainDistribution(address account) external view returns (address) {
    DistributionModuleStorageV0 storage $ = _distributionModuleStorageV0();
    return $.initiatedRedirectedOffChainDistribution[account];
}
```

Usual: Fixed in PR 2219.

Cantina Managed: Fix verified.

3.4.2 `DistributionModule: feeAmount` for ERC20 `$.usual.safeTransfer` can be zero

Severity: Informational

Context: DistributionModule.sol#L1014

Description: The `DistributionModule` distributes fees based on basis points.

```
feeAmount = Math.mulDiv(feeSwept, $.usualXFeeRate, BASIS_POINT_BASE, Math.Rounding.Floor);
```

The resulting actual `feeAmount` can be zero. Either because the `feeRate` is zero basis points or the `feeSwept` (totalFeeAmount) is too small. The ERC20 tranfer would be called with `feeAmount=0` which would be a no-op. All three fee receivers `UsualX`, `treasuryYield` and burn have the same problem.

Recommendation: Consider to only call the erc20 `safeTransfer` or burn if the `feeAmount` is higher than zero. Otherwise, it would be a no-op.

```
if (feeAmount > 0) {
   $.usual.safeTransfer(address($.usualX), feeAmount);
}
```

Usual: Fixed in PR 2234.

Cantina Managed: Fix verified.

3.4.3 General improvements like code consistency, naming and missing natspec

Severity: Informational

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

Description:

- **Missing Natspec:** The `DistributionModule._distributeFeesToUsualX` function is missing a `natspec` like other similar functions (`_distributeFeesToYieldTreasury`, `_burnFees`).
- **Variable Namings:**
  - Some variables are very long lik `initiatedRedirectedOffChainDistributionStartingTimestamp`.
  - the event name `UsualFeeAllocatedToBurn` doesn't make sense, since the fees are already burned. A name like `UsualFeeBurned` would be more descriptive.
  - `UsualX.sweepFees` and `USD0PP.sweepFees` could use the same helper variable names since both functions execute the same logic.

Function specifiers: `DistributionModule.calculateUsualDist` can be marked as `external`.

- **Gas optimizations for new storage variables:** Multiple struct member variables in `DistributionModuleStorageV0` don't need to be `uint256` However, this can't be changed in retrospective because it would break the storage. However, newly introduced variables like `treasuryFeeRate, usualXFeeRate, burnFeeRate` could be `uint16` since the max value is `10_000`.

Recommendation: Consider the list above as optional improvements for the codebase.

Usual: Fixed in PR 2236.

Cantina Managed: Fix verified.

3.4.4 Reset `unused*` state variables of contracts during migration

Severity: Informational

Context: Usd0PP.sol#L122, UsualX.sol#L95

Description: The `Usd0PP.Usd0PPStorageV0.unusedTreasuryAllocationRate` and `UsualX.UsualXStorageV0.unusedBurnRatioBps` are storage state variables which are used in the current onchain contracts but won't be used anymore after the migration to new contract implementations. They are just kept there to preserve the storage layout of contracts.

Ideally since those storage values won't be used anymore, they should be reset to `0` during or before the migration. Keeping unused non-zero storage values are not ideal and can cause issues in future implementations of contracts if not handled carefully.

Recommendation: Consider resetting `Usd0PP.Usd0PPStorageV0.unusedTreasuryAllocationRate` and `UsualX.UsualXStorageV0.unusedBurnRatioBps` are storage state variables to `0` values during migration.

Usual: Acknowledged. Will be handled carefully (e.g. our namespace storaged struct is append-only).

Cantina Managed: Acknowledged.