CANTINA

Usual Pegasus
Security Review

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

June 10, 2024

Contents

1 Introduction **2**
- 1.1 About Cantina 2
- 1.2 Disclaimer 2
- 1.3 Risk assessment 2
  - 1.3.1 Severity Classification 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Medium Risk 4
  - 3.1.1 `usd0.blacklist` allows to blacklist `address(0)` which would halt the protocol 4
  - 3.1.2 `usd0pp.emergencywithdraw` event doesn't pause `mint` and `unwrap`. 4
  - 3.1.3 Possible to `mint` usd0pp tokens before `bondstart` 4
  - 3.1.4 Permit call within `provideusd0receiveusdcwithpermit` will almost certainly fail 5
  - 3.1.5 Attacker can DoS all `swaprwatostbcintent` calls 5
- 3.2 Low Risk 6
  - 3.2.1 `intent_type_hash` is declared incorrectly 6
  - 3.2.2 Blacklisted or not whitelisted user can hold usd0pp tokens 6
  - 3.2.3 A partially filled usdc order might be avoided by `swapperengine.provideusd0receiveusdc` callers 6
  - 3.2.4 `daocollateral` has no `invalidnonce` function for `intents` 7
  - 3.2.5 Missing `amountintoken > type(uint128).max` check in `daocollateral._swaprwatostbc` 7
  - 3.2.6 No `registryaccess.onlymatchingrole(allowlisted)` check in the `swaprwatostbc` functions compared to swap and redeem 7
  - 3.2.7 Users who are blacklisted/not allowlisted can fill `swapperengine` with unfillable orders 7
  - 3.2.8 `daocollateral#_swaprwatostbc` will leave a few wei of usd0 stuck within the contract 8
- 3.3 Gas Optimization 8
  - 3.3.1 `swapperengine` could minimize the erc20 transfers of `usd0` and `usdc` 8
  - 3.3.2 The price from the oracle could be cached inside the loop in `swapperengine._provideusd0receiveusdc` 8
  - 3.3.3 The redundant set of `unmatchedusd0inwad` in the `else` clause can be removed 9
  - 3.3.4 Skip external `permit()` call to save gas 9
- 3.4 Informational 9
  - 3.4.1 `swapperEngine` assumes a `1$ == 1 usd0`, a depeg would allow arbitrage and results in losses for usdc depositors 9
  - 3.4.2 No `maxusdcprice` for buyer's and no `minusdcprice` for sellers in `swapperengine` 9
  - 3.4.3 Lack of events for `usd0.blacklist` and usd0.unblacklist' 10
  - 3.4.4 Different implementation for `swapperengine.swapusd0` avoiding `dust` 10
  - 3.4.5 Remove all `TODO`s 10
  - 3.4.6 Unused constants in `constants.sol` 11
  - 3.4.7 If `$.usd0.balanceof(msg.sender) < requiredusd0amount` check only happens in `provideusd0receiveusdcwithpermit` 11
  - 3.4.8 Consider an amount parameter for `usd0pp.unwrap` 11
  - 3.4.9 Incorrect/incomplete natspec 11
  - 3.4.10 `emergencywithdraw` does unnecessary `address(0)` check 12

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 May 17th to May 22nd the Cantina team conducted a review of pegasus-solidity on commit hash 3d1b7406. The team identified a total of **27** issues in the following risk categories:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	5	4	1
Low Risk	8	7	1
Gas Optimizations	4	4	0
Informational	10	7	3
Total	**27**	**22**	**5**

3 Findings

3.1 Medium Risk

3.1.1 `usd0.blacklist` allows to blacklist `address(0)` which would halt the protocol

Severity: Medium Risk

Context: Usd0.sol#L194

Description: The `blacklist` function in the `USD0` token contract allows to `blacklist` addresses. A `blacklisted` address cannot use `USD0`. There is a missing check in the blacklist function:

```
if (account == address(0)) {
  revert InvalidAddress();
}
```

If the `address(0)` is blacklisted it would be not possible to `mint` or `burn` USD0 tokens. This could be especially problematic if different `roles` are used to manage the `blacklisting` and other overall `admin` functionality like `pausing` the protocol. The `blacklist` role would get indirect the power to halt the protocol.

Recommendation: Add the following check to the `blacklist` function:

```
if (account == address(0)) {
  revert InvalidAddress();
}
```

Usual: Fixed in PR 1016.

Cantina Managed: Fixed.

3.1.2 `usd0pp.emergencywithdraw` event doesn't pause `mint` and `unwrap`.

Severity: Medium Risk

Context: Usd0PP.sol#L193

Description: The `USD0PP` contract has an emergency function called `emergencyWithdraw` which allows withdrawing all `USD0` tokens. In case of an `emergencyWithdraw`, it would still be possible to mint new bond tokens or call `unwrap` after the bond duration is finished.

The `unwrap` function would exchange bond tokens for `USD0`. After an `emergencyWithdraw` event, the contract can still accumulate new `USD0` tokens with `mint`, but this would result in a first-come, first-served situation for `unwrap` after the bond has finished.

Recommendation: Consider adding a `pause` functionality to the contract or `pause` the `mint` function in case of an `emergencyWithdraw` event.

Usual: Fixed in PR 1046.

Cantina Managed: Fixed.

3.1.3 Possible to `mint` usd0pp tokens before `bondstart`

Severity: Medium Risk

Context: Usd0PP.sol#L143

Description: The `USD0PP` contract is an implementation of a four-year `bond` for `USD0`. In the `constructor`, a `bondStart` timestamp parameter defines the start of the `bond`. The `bondStart` is required to be a timestamp in the future. However, the `mint` function is missing a check to ensure tokens can only be minted when the bond has started.

Recommendation: Add the following check to the `mint` function:

```
if (block.timestamp < $.bondStart) {
revert BondNotStarted();
}
```

Usual: Fixed in PR 1021.

Cantina Managed: Fixed.

3.1.4 Permit call within `provideusd0receiveusdcwithpermit` will almost certainly fail

Severity: Medium Risk

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

Description: Within `provideUsd0ReceiveUSDCWithPermit`, the user specifies the amount of `USDC` they want to take and based on that amount a `permit` call is made.

```
uint256 requiredUsd0Amount =
    _getUsd0WadEquivalent(amountUsdcToTakeInNativeDecimals, usdcWadPrice);
// Authorization transfer
if ($.usd0.balanceOf(msg.sender) < requiredUsd0Amount) {
    revert InsufficientUSD0Balance();
}
try IERC20Permit(address($.usd0)).permit(
    msg.sender, address(this), requiredUsd0Amount, deadline, v, r, s
) {} catch {} // s
```

The problem is that `amountUsdcToTakeInNativeDecimals` is dynamically calculated and is subject to changes and in order for the permit to work, it would need `amountUsdcToTakeInNativeDecimals` to exactly match the `amount` from the signature.

Recommendation: Allow users to specify the exact amount of funds they'll permit.

Usual: Fixed in PR 1071.

Cantina Managed: Fixed.

3.1.5 Attacker can DoS all `swaprwatostbcintent` calls

Severity: Medium Risk

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

Description: Within `DaoCollateral`, users can use signatures (`Intent`) to authorize admins/ other users to execute `_swapRWAtoStbc` on their behalf (swap RWA for USDC). The problem is that the provided `Intent` signature neither includes `partialMatching` parameter, nor a minimum amount of USDC to be taken.

This, combined with the fact that `swapRWAtoStbcIntent` is permisionless, allows any user to front-run a call to `swapRWAtoStbcIntent` and use the provided intent to only take a dust position (using `partialMatching == true`). This would later make the honest transaction revert, as the nonce provided in the `Intent` will have already been used.

Recommendation: Add `partialMatching` as part of the signature. Consider making `swapRWAtoStbcIntent` a restricted function.

Usual: Acknowledged. Calling the `DaoCollateral` requires being `ALLOWLISTED`. If an allowlisted address starts to call `swapRWAtoStbcIntent` continuously to the disadvantage of the USDC buyer, the address will be removed.

Cantina Managed: Acknowledged.

3.2 Low Risk

3.2.1 `intent_type_hash` is declared incorrectly

Severity: Low Risk

Context: DaoCollateral.sol#L754

Description: In the EIP-712 specification, the hash struct that is signed refers to the struct type. However, the `INTENT_TYPE_HASH` is currently declared as `keccak256("swapRWAtoStbcIntent(uint256[],Approval,Intent,bool)")`, which refers to the invoked function, not the signed struct. To ensure compatibility with `EIP-712`, this difference needs to be fixed.

Recommendation: The recommendation is to separate the signature from the `Intent` struct and use then use the `Intent` as the `INTENT_TYPE_HASH`:

```
- bytes32 constant INTENT_TYPE_HASH = keccak256("swapRWAtoStbcIntent(uint256[],Approval,Intent,bool)");
+ bytes32 constant INTENT_TYPE_HASH = keccak256("SwapIntent(address recipient,address rwaToken,uint256
↪   amountInTokenDecimals,uint256 nonce,uint256 deadline)")`
```

Usual: Fixed in PR 1064.

Cantina Managed: Fixed.

3.2.2 Blacklisted or not whitelisted user can hold usd0pp tokens

Severity: Low Risk

Context: Usd0PP.sol#L257

Description: The `Usd0` contract implements a `blacklist` and `whitelist`. To transfer tokens, a user must be on the `whitelist` and not on the `blacklist`. However, the `Usd0PP` contract does not perform this check before transferring tokens. This allows any user to transfer `Usd0PP` tokens to a user who is not whitelisted or is blacklisted.

While this does not have any impact because a malicious user cannot redeem their `Usd0` tokens, adding these conditions would make the contracts more consistent.

Recommendation: It's recommended to check if a user is already blacklisted or not whitelisted before transferring tokens.

Usual: Fixed in PR PR 1045.

Cantina Managed: Fixed.

3.2.3 A partially filled usdc order might be avoided by `swapperengine.provideusd0receiveusdc` callers

Severity: Low Risk

Context: SwapperEngine.sol#L140

Description: If a user deposits `USDC` to the `swapperEngine`, the amount needs to be at least the `minimumUSDCAmountProvided`. However, an order can be partially fulfilled. The remaining amount might not be worth filling for other `USDC` buyers to reduce gas costs.

This situation would force the `USDC` depositor to withdraw their order from a long-term perspective.

Recommendation: This implication most likely needs to be accepted with the current design. The only solution would be to require the `provideUsd0ReceiveUSDC` to buy `USDC` in multiples of a certain factor. For example, enforce an amount increase in 100 USDC steps.

Usual: Acknowledged. Users need to be `allowlisted` to use `USD0` which prohibits malicious gas-wasting on orderTaking by leaving small amounts. Most users will use the app provided by the Usual team, which will include orders with a lower amount, and our intent-system will closely monitor & prioritize full order completion.

Cantina Managed: Acknowledged.

3.2.4 `daocollateral` has no `invalidnonce` function for `intents`

Severity: Low Risk

Context: DaoCollateral.sol#L749

Description: There is no `invalidNonce` function in the `DAOCollateral` contract. It would be not possible to cancel a `DaoCollateral.swapRWAtoStbcIntent` by the message signer.

Recommendation: Add a `invalidNonce` function to the `DaoCollateral` contract to `cancel` a signature. The function would increase the `nonce` counter for the `msg.sender`.

Usual: Fixed in PR 1075.

Cantina Managed: Fixed.

3.2.5 Missing `amountintoken > type(uint128).max` check in `daocollateral._swaprwatostbc`

Severity: Low Risk

Context: DaoCollateral.sol#L576

Description: The `_swapRWAtoStbc` function doesn't have an `amountInToken > type(uint128).max` check like the `swap` function. This check should be added since `swapRWAtoStbc` performs operations similar to those in `DaoCollateral.swap`.

Recommendation: Add the same safety checks to the `_swapRWAtoStbc` as for the `swap` function.

Usual: Fixed in PR 1036.

Cantina Managed: Fixed.

3.2.6 No `registryaccess.onlymatchingrole(allowlisted)` check in the `swaprwatostbc` functions compared to swap and redeem

Severity: Low Risk

Context: DaoCollateral.sol#L723

Description: There is no `registryAccess.onlyMatchingRole(ALLOWLISTED)` check for the `daoCollateral.swapRWAtoStbc` functions. Since the `swapRWAtoStbc` function performs operations internally similar to `swap` and `redeem`, this check should be added.

Recommendation: Consider if overall the `registryAccess.onlyMatchingRole(ALLOWLISTED)` check is required for the `swap` and `redeem` functions. If so, the `swapRWAtoStbc` function should include it as well.

Usual: Fixed in PR PR 1050.

Cantina Managed: Fixed.

3.2.7 Users who are blacklisted/not allowlisted can fill `swapperengine` with unfillable orders

Severity: Low Risk

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

Description: In order for a user to be able to receive `Usd0`, they'd need to be allowlisted/ not blacklisted. In case they don't satisfy any of the criteria, they won't be able to receive it. However, this would not stop them from creating orders within `SwapperEngine` via `depositUSDC`.

Only when an honest user attempts to fill their order, the `Usd0` transfer will fail, halting the execution of the transaction. This could cause many user transactions to unexpectedly revert while in the middle of filling multiple orders, effectively resulting in loss of funds in terms of the failed transaction's gas cost.

Recommendation: Do not allow users who cannot receive `Usd0` to create orders within `SwapperEngine`.

Usual: Fixed in PR 1054.

Cantina Managed: Fixed.

3.2.8 `daocollateral#_swaprwatostbc` will leave a few wei of usd0 stuck within the contract

Severity: Low Risk

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

Description: When `swapperEngine#swapUsd0` is called within `_swapRWAtoStbc`, the returned value is trusted to be the exact amount of `Usd0` that is not used. However, due to rounding downs, this is not the case:

```
uint256 dust = amountUsd0ToProvideInWad
    - _getUsd0WadEquivalent(amountUsdcToTakeInNativeDecimals, usdcWadPrice);
```

When calculating the unused dust amount, the contract wrongfully assumes that `_getUsd0WadEquivalent(amountUsdcToTakeInNativeDecimals, usdcWadPrice);` will return the actual swapped Usd0 amount. This is based on the assumption that `_getUsd0WadEquivalent(x, usdcWadPrice)` + _getUsd0WadEquivalent(y, usdcWadPrice);=_getUsd0WadEquivalent(x+y, usdcWadPrice);‘.

However, since `_getUsd0WadEquivalent` rounds down the result, it could cause the left side of the equation above to be 1 wei less than the right side. Every wei difference caused due to the amount of different of `orderIds` taken will remain stuck within `DaoCollateral`.

Recommendation: Instead of calculating how much should the unused amount be, simply track the balance difference of the `msg.sender`.

Usual: Fixed in PR 1025.

Cantina Managed: Fixed.

3.3 Gas Optimization

3.3.1 `swapperengine` could minimize the erc20 transfers of `usd0` and `usdc`

Severity: Gas Optimization

Context: SwapperEngine.sol#L315

Description: In the current `SwapperEngine` design, each order match results in two ERC20 transfers inside the `_provideUsd0ReceiveUSDC` function.

Recommendation: The transfer of USDC from the contract to the `recipient` could happen once with the total sum outside of the loop. Instead of using a `push` pattern for the `USD0` transfers, the orders could be marked as `fulfilled`, and the depositor would have to call the contract again to receive the `USD0` (pull pattern).

The depositor could provide an array of different `orderId, resulting in one` USD0`transfer. However, this would require transferring the`USD0`amounts first to the contract inside the`_provideUsd0ReceiveUSDC‘ function.

Usual: Fixed in PR 1067

Cantina Managed: Fixed.

3.3.2 The price from the oracle could be cached inside the loop in `swapperengine._provideusd0receiveusdc`

Severity: Gas Optimization

Context: SwapperEngine.sol#L313

Description: In `SwapperEngine._provideUsd0ReceiveUSDC` the price from the oracle could be cached instead of calling the oracle each time inside of the loop.

Recommendation: Cache the price of `USDC` in a local variable and avoid calling the oracle contract each time inside the loop.

Usual: Fixed in PR 1077.

Cantina Managed: Fixed.

3.3.3 The redundant set of `unmatchedusd0inwad` in the `else` clause can be removed

Severity: Gas Optimization

Context: SwapperEngine.sol#L396-L398

Description: The variable `unmatchedUsd0InWad` is already set to `0`, so there's no need to reset it in the `else` clause.

Recommendation: The recommendation is to remove the `else` clause.

Usual: Fixed in PR 1037.

Cantina Managed: Fixed.

3.3.4 Skip external `permit()` call to save gas

Severity: Gas Optimization

Context: DaoCollateral.sol#L587

Description: A user may have already added approval before calling the function and setting the Approval parameter to null in certain cases. When they then call the `swapRWAtoStbc()` function, there is no need to call the `permit()` function. Skipping the `permit()` saves gas.

Recommendation: It's recommended to skip the `permit()` call when it's not set.

Usual: Fixed in PR 1062.

Cantina Managed: Fixed.

3.4 Informational

3.4.1 `swapperEngine` assumes a `1$ == 1 usd0`, a depeg would allow arbitrage and results in losses for usdc depositors

Severity: Informational

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

Description: The current `SwapperEngine` assumes `1 USD0 == 1 USD`. If the price of `1 USD0` falls below `1 USD`, this would open an arbitrage opportunity. The arbitrager would buy `USD0` at a cheaper price and take all the USDC deposits in the `SwapperEngine` for profit.

Recommendation: This assumption needs to be documented, and USDC depositors need to be made aware of the risk.

Usual: Acknowledged. This assumption is safeguarded by several mechanisms, such as price-depeg checks, the CBR mechanism, pausability, offchain monitoring and our own routing mechanism preventing USDC deposits for an unfavorable trade if other DeFi solutions offer better rates (i.e. curvepool/uniswap).

Cantina Managed: Acknowledged.

3.4.2 No `maxusdcprice` for buyer's and no `minusdcprice` for sellers in `swapperengine`

Severity: Informational

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

Description: The `SwapperEngine` has no option to define a `maxUSDCPrice` for buyers. Similarly, sellers do not have the option to define a `minimumUSDCPrice`. The actual price is provided by a USDC oracle. Since the `buyer` is the `taker` by calling `provideUsd0ReceiveUSDC`, they could optimize returns by performing trades at `low` USDC prices. On the other hand, sellers have some control because they could trigger a buyer `Intent` at a `high` USDC price by calling `DaoCollateral.swapRWAtoStbcIntent`.

Recommendation: This behavior needs to be documented and users should be aware of the risk.

Usual: Acknowledged.

Cantina Managed: Acknowledged.

3.4.3 Lack of events for `usd0.blacklist` and usd0.unblacklist‘

Severity: Informational

Context: Usd0.sol#L188

Description: Currently, the `blacklist` functions `blacklist` and `unBlacklist` don't use events.

Recommendation: Add an event for `Blacklist` and `Unblacklist`.

Usual: Fixed in PR 1016.

Cantina Managed: Fixed.

3.4.4 Different implementation for `swapperengine.swapusd0` avoiding `dust`

Severity: Informational

Context: SwapperEngine.sol#L378

Description: Instead of calculating the conversion between `USD0` and `USDC` which can result in `dust` amounts. The calculation could be based on `usd0.balanceOf` to be precise.

Recommendation: `swapUsd0` function based on `usd0.balanceOf`:

```
function swapUsd0(
    address recipient,
    uint256 amountUsd0ToProvideInWad,
    uint256[] memory orderIdsToTake,
    bool partialMatchingAllowed
) external nonReentrant returns (uint256) {
    uint256 usdcWadPrice = _getUsdcWadPrice();
    SwapperEngineStorageV0 storage $ = _swapperEngineStorageV0();

    uint256 preUSD0Balance = $.usd0.balanceOf(address(msg.sender));

    uint256 unmatched = _provideUsd0ReceiveUSDC(
        recipient, _getUsdcAmountFromUsd0WadEquivalent(amountUsd0ToProvideInWad, usdcWadPrice),
↪   orderIdsToTake, partialMatchingAllowed
    );

    if (unmatched == 0) {
        return 0;
    }

    return amountUsd0ToProvideInWad - (preUSD0Balance - $.usd0.balanceOf(address(msg.sender)));
}
```

Alternatively, the `_provideUsd0ReceiveUSDC` function could return the total taken `usd0` amount.

Usual: Fixed in PR 1025.

Cantina Managed: Fixed.

3.4.5 Remove all `TODO`s

Severity: Informational

Context: DaoCollateral.sol#L616

Description: As a best practice, the code to be deployed should not contain any `TODO` comments.

Recommendation: The recommendation is to remove or implement all `TODO`s

Usual: Fixed in PR 1035.

Cantina Managed: Fixed.

3.4.6 Unused constants in `constants.sol`

Severity: Informational

Context: constants.sol#L43

Description: Multiple constants are not used in the current version like `CANCEL_FEE`, `MAX_CANCEL_FEE`, `WAD_MINIMUM_RWA_CONSTRUCTOR`, etc..

Recommendation: Remove unused constants or constants only used in tests from the `constants.sol`.

Usual: Fixed in PR 1034.

Cantina Managed: Fixed.

3.4.7 If `$.usd0.balanceof(msg.sender) < requiredusd0amount` check only happens in `provideusd0receiveusdcwithpermit`

Severity: Informational

Context: SwapperEngine.sol#L366

Description: The check `if ($.usd0.balanceOf(msg.sender) < requiredUsd0Amount)` then `revert`, is an early revert condition for the `provideUsd0ReceiveUSDCWithPermit` function. In the regular `provideUsd0ReceiveUSDC` function such a check doesn't exist.

Recommendation: This check is not specific to the `provideUsd0ReceiveUSDCWithPermit` and could be used in `provideUsd0ReceiveUSDC` as well.

Usual: Fixed in PR 1047.

Cantina Managed: Fixed.

3.4.8 Consider an amount parameter for `usd0pp.unwrap`

Severity: Informational

Context: Usd0PP.sol#L193

Description: Currently, there is no `amount` parameter in the `unwrap` function. Currently a `unwrap` call would use the entire `balanceOf` bond tokens of the `msg.sender`.

Recommendation: Consider adding an amount parameter to `USD0PP.unwrap` to allow users more flexibility.

Usual: Acknowledged.

Cantina Managed: Acknowledged.

3.4.9 Incorrect/incomplete natspec

Severity: Informational

Context: ISwapperEngine.sol#L41, ISwapperEngine.sol#L60

Description: There are some case where the NatSpec is incorrect:

1. ISwapperEngine.sol#L42: Missing the @return statement
2. ISwapperEngine.sol#L60: The @return statement is incorrect, it returns the unmatched amount of USD0 in WAD.

Recommendation: It's recommended to fix the NatSpec.

Usual: Fixed in PR 1027.

Cantina Managed: Fixed.

3.4.10 `emergencywithdraw` does unnecessary `address(0)` check

Severity: Informational

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

Description:

```
function emergencyWithdraw(address safeAccount) external {
    Usd0PPStorageV0 storage $ = _usd0ppStorageV0();

    if (!$.registryAccess.hasRole(DEFAULT_ADMIN_ROLE, msg.sender)) {
        revert NotAuthorized();
    }
    if (safeAccount == address(0)) {
        revert NullAddress();
    }
    IERC20 usd0 = $.usd0;

    uint256 balance = usd0.balanceOf(address(this));
    // get the collateral token for the bond
    usd0.safeTransfer(safeAccount, balance);

    emit EmergencyWithdraw(safeAccount, balance);
}
```

Here, the contract does a check that the provided `safeAccount` is not `address(0)` in order to avoid mistakenly sending the funds to it. However, that is unnecessary as `usd0` is an OZ ERC20 and it will revert on a transfer to `address(0)`.

Recommendation: Remove the `address(0)` check.

Usual: Fixed in PR 1021.

Cantina Managed: Fixed.