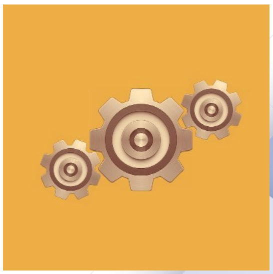
Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR



Cog Isolated Lending Platform

Veridise Inc.
August 24, 2023

► Prepared For:

Cog-Finance
`https://www.cog.finance`

► Prepared By:

Ajinkya D. Rajput
Timothy Hoffman

► Contact Us: `contact@veridise.com`

► Version History:

Thu. 24, Aug 2023	V1
Mon. 14, Aug 2023	Initial Draft

Contents

Executive Summary 1

From July. 31, 2023 to Aug. 7, 2023, Cog-Finance engaged Veridise to review the security of their Cog Isolated Lending Platform. Cog is an isolated lending protocol, focused towards decentralization and capital efficiency. Cog allows users to deploy permissionless isolated lending/borrowing pools. This is the first audit of Cog Isolated Lending Platform performed by Veridise. Veridise conducted the assessment over 12 person-days, with 2 engineers reviewing code over 6 days from commit `7ca2a6a`. The auditing strategy involved manual auditing by engineers.

Code assessment. The Cog Isolated Lending Platform developers provided the source code of the Cog Isolated Lending Platform contracts for review. To facilitate the Veridise auditors' understanding of the code, the Cog Isolated Lending Platform developers also provided link to their documentation. The source code also contained detailed documentation in the form of READMEs and documentation comments on functions and storage variables.

The source code contained a test suite, which the Veridise auditors noted had tests for some use cases. However, test cases were missing for some use cases like withdrawal and borrowing. The test suite also included stateful tests to check if the protocol maintains a consistent internal state throughout a series of transactions. Stateful tests are useful for modeling real-world scenarios and ensuring that the protocol behaves as expected on both valid and invalid inputs.

Summary of issues detected. The audit uncovered 18 issues, 6 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, V-COG-VUL-001 and V-COG-VUL-002 are critical logic issues that allow an attacker to steal assets from borrowing pool and collateral from users respectively. V-COG-VUL-003 is a high severity issue which finds that the surge protection described in the documentation is partially implemented in code. V-COG-VUL-004 is another high severity issue where the redeem function transfers tokens to the caller but the return value and log give an incorrect number of assets. V-COG-VUL-005 and V-COG-VUL-006 are other high severity issues where the protocol transfer fewer tokens than needed in repay and liquidate functions respectively. The Veridise auditors also identified several medium-severity issues, including V-COG-VUL-006 where the withdraw function transfers correct amount of tokens but reports a lesser number of tokens as transferred. Also, V-COG-VUL-007 finds a comparison between quantities of different units. There were also a number of low severity issues and warnings. The Cog Isolated Lending Platform developers acknowledged all of the reported issues and fixed 16 out of 18 issues.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the Cog Isolated Lending Platform.
Interaction of units The protocol deals with two different units for asset tokens deposited in the liquidity pool, i.e. the absolute number of tokens and shares. Correct inter-conversion of these quantities is critical and needs to be tested thoroughly.
Testing The test suite was missing test cases for a few use cases and we recommend to add

test cases for missing use cases. We also recommend testing scenarios where longer sequences of user actions are performed therefore testing interaction of different use cases, especially sequences that trigger interest calculation.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

Project Dashboard 2

Table 2.1: Application Summary.

Name	Version	Type	Platform
Cog Isolated Lending Platform	`7ca2a6a`	Vyper	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
July. 31 - Aug. 7, 2023	Manual	2	12 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	2	2
High-Severity Issues	4	4
Medium-Severity Issues	2	2
Low-Severity Issues	3	3
Warning-Severity Issues	6	6
Informational-Severity Issues	1	1
TOTAL	18	18

Table 2.4: Category Breakdown.

Name	Number
Logic Error	11
Maintainability	5
Data Validation	1
Gas Optimization	1

Audit Goals and Scope 3

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Cog Isolated Lending Platform's smart contracts. In our audit, we sought to answer the following questions:

- Can attacker steal asset tokens from liquidity pool?
- Can attacker steal collateral tokens from users?
- Are arithmetic operations safe?
- Does protocol transfer right amount of tokens?
- Does protocol report right amount of tokens transferred?
- Does the protocol employ right access control?
- Does the protocol interact correctly with oracles?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved human experts manually auditing issues.
Scope. The scope of this audit is limited to the `src` folder of the source code provided by the Cog Isolated Lending Platform developers, which contains the smart contract implementation of the Cog Isolated Lending Platform, specifically,

- `src/cog_factory.vy`
- `src/cog_pair.vy`
- `src/fuse_box.vy`
- `src/loan_router.vy`

Methodology. Veridise auditors read the Cog Isolated Lending Platform documentation and inspected the provided tests. They then performed a manual audit of the code. During the audit, the Veridise auditors regularly met with the Cog Isolated Lending Platform developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

Table 3.4: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-COG-VUL-001	Attacker may steal all assets of cog_pair	Critical	Fixed
V-COG-VUL-002	Attacker can steal collateral from arbitrary user	Critical	Fixed
V-COG-VUL-003	Interest rate surge Protection are not implemented	High	Fixed
V-COG-VUL-004	Redeem returns wrong number of assets transferre	High	Fixed
V-COG-VUL-005	Protocol transfers in fewer funds in repay()	High	Fixed
V-COG-VUL-006	Protocol transfers in fewer funds in liquidate()	High	Fixed
V-COG-VUL-007	Wrong amount returns as shares from withdraw	Medium	Fixed
V-COG-VUL-008	Comparison of shares and ERC20 tokens	Medium	Fixed
V-COG-VUL-009	Possible overflow while calculating mean price	Low	Fixed
V-COG-VUL-010	Consider using ‘mul_div’ in more locations	Low	Won’t Fix
V-COG-VUL-011	Subtracting values of different units	Low	Fixed
V-COG-VUL-012	Check if atleast one oracle is active in fuse_box	Warning	Fixed
V-COG-VUL-013	Unnecessary memory copy	Warning	Won’t Fix
V-COG-VUL-014	Divide before multiply can give incorrect 0 result	Warning	Fixed
V-COG-VUL-015	_isPaused() function name is confusing	Warning	Fixed
V-COG-VUL-016	Unused variable or dead code	Warning	Fixed
V-COG-VUL-017	Use of magic number	Warning	Fixed
V-COG-VUL-018	Inconsistent or missing documentation	Info	Fixed

3.4 Detailed Description of Issues

3.4.1 V-COG-VUL-001: Attacker may steal all assets of cog_pair

Severity	Critical	**Commit**	7ca2a6a
Type	Logic Error	**Status**	Fixed
File(s)	`src/cog_pair.vy`		
Location(s)	borrow()		

The protocol allows users to borrow `asset` tokens against the `collateral` deposited by them by calling the `borrow()` external function. The protocol also provides a `borrow_approvals` mechanism for users to set allowances for other users to borrow on their behalf against the `collateral` deposited by them.

```
@external
def approve_borrow(borrower: address, amount: uint256) -> bool:
    self.borrow_approvals[msg.sender][borrower] = amount
    log Approval(msg.sender, borrower, amount)
    return True
```

Figure 3.1: `approve_borrow()` in cog_pair.vy

A user `A` can allow another user `B` to borrow `amount` funds on behalf of `A` against collateral deposited by `A` by calling `approve_borrow(B, amount)`

```
def borrow(
    amount: uint256, _from: address = msg.sender, to: address = msg.sender
) -> uint256:
    """
    @param to The address to send the borrowed tokens to
    @param amount The amount of asset to borrow, in tokens
    @return The amount of tokens borrowed
    """
    self._isPaused()
    self.efficient_accrue()
    if _from != msg.sender:
        self.borrow_approvals[_from][msg.sender] -= amount
    borrowed: uint256 = self._borrow(amount, _from, to)
    assert self._is_solvent(
        msg.sender, self.exchange_rate
    ), "Insufficient Collateral"
    accrue_info: AccrueInfo = self.accrue_info
    # Now that utilization has changed, interest must be accrued to trigger any surge which now may be occuring
    self._accrue(self.accrue_info, 0)
    return borrowed
```

Figure 3.2: `borrow()` in cog_pair.vy

The `borrow()` function takes 3 arguments:

- `amount`: The amount to be borrowed.

- `_from = msg.sender`: The address on behalf of which the borrow is requested (default value is `msg.sender`).
- `_to = msg.sender`: The address to which the borrowed tokens are to be transferred.

The `borrow()` function performs the following:

1. Accrue interest by calling `self.efficient_accrue()`.
2. Checks if `_from` is not the default value of `msg.sender`, and if so, reduces approval for `_from` provided by `msg.sender` represented by `self.borrow_approvals[_from][msg.sender]`.
3. Call internal `_borrow()` function.
4. Assert if the account is solvent: `self._is_solvent(msg.sender, self.exchange_rate)`.
5. Accrue interest by calling `_accrue()`.
6. Return borrowed tokens.

The borrow is performed on behalf of `_from` and solvency is checked for the `msg.sender`. When `_from` is not equal to default value of `msg.sender`, the protocol will allow an attacker to borrow funds on behalf of an insolvent account.

Impact An attacker can steal all the assets in the liquidity pool of the pair.

Consider the following attack scenario:

1. Attacker uses two addresses `Driver` and `Borrower`.
2. Attacker calls `approve_borrow(Borrower, max_value(uint256))` from `Driver` address.
3. Attack calls `borrow(<all_assets>, Driver)` from `Borrower` address.
    a) This call will go through because
        i. The borrow will be made from `Driver` account.
        ii. Solvency will be checked for `Borrower` account.
        iii. Since `Borrower` has zero borrow at this point, it will be assessed to be solvent and the assert will pass.
    b) This would transfer all assets in the pair liquidity pool to `Borrower`.
4. The attacker does not have any collateral invested in the protocol so the attacker has no compulsion to repay the borrow.

Recommendation The `_is_solvent` assertion at `cog_pair.vy:1166` should check `_from` instead of `msg.sender`.

Developer Response Developers have acknowledged and fixed this issue.

3.4.2 V-COG-VUL-002: Attacker can steal collateral from arbitrary user

Severity	Critical	**Commit**	7ca2a6a
Type	Logic Error	**Status**	Fixed
File(s)	`src/cog_pair.vy`		
Location(s)	_remove_collateral()		

The protocol allows users to deposit and withdraw collateral and borrow the asset tokens against the deposited collateral. The protocol maintains the balance of deposited collateral of an user in `self.user_collateral_share` map. Users can withdraw their collateral using `remove_collateral()` function. These functions call an internal function `_remove_collateral()` which implements the steps for bookkeeping and transferring the tokens to the withdrawer.

```
@internal
def _remove_collateral(to: address, amount: uint256):
    """
    @param to The address to remove collateral for
    @param amount The amount of collateral to remove, in tokens
    """
    new_collateral_share: uint256 = self.user_collateral_share[to] - amount
    self.user_collateral_share[msg.sender] = new_collateral_share
    self.total_collateral_share = self.total_collateral_share - amount
    assert ERC20(collateral).transfer(
        to, amount, default_return_value=True
    )  # dev: Transfer Failed

    log RemoveCollateral(to, amount, new_collateral_share)
```

Figure 3.3: `_remove_collateral()` in `cog_pair.vy`

The function takes two arguments:

- `to`: the destination address for token transfer
- `amount`: the number of shares to be withdrawn

In the first step, the function calculates the new collateral balance of the user in `new_collateral_share` after withdrawing `amount` by subtracting `amount` from `self.user_collateral_share[to]` and updating the balance of `msg.sender` with `new_collateral_share`.

```
new_collateral_share: uint256 = self.user_collateral_share[to] - amount
```

This calculates an incorrect balance of the user after collateral withdrawal because it takes the old collateral balance of `to` instead of `msg.sender` and updates the collateral balance of `msg.sender` with the incorrect balance.

Impact An attacker can withdraw funds from any arbitrary account.

Attack scenario:

1. Consider victims initial collateral balance to be **B**
2. An attacker deposits a small amount, **X**, of collateral in the protocol.

3. An attacker the requests to withdraw **X** collateral with address `to` set to victim. This will update the attacker's balance to **B-X**.
4. Now attacker can perform another call to `remove_collateral()` for amount **B-X** with `to` set to himself. This will remove all of the victims collateral to the attacker and this call will be successful because the attacker is solvent as he does not have any outstanding borrow.

Recommendation `new_collateral_share` should be calculated as

```
self.user_collateral_share[msg.sender] - amount
```

Developer Response Developers have acknowledged and fixed this issue.

3.4.3 V-COG-VUL-003: Interest rate surge Protection are not implemented

Severity	High	**Commit**	7ca2a6a
Type	Logic Error	**Status**	Fixed
File(s)	`src/cog_pair.vy`		
Location(s)	_accrue()		

The protocol allows for a dynamic interest rate based on utilization of the liquidity pool. The protocol decreases the interest rate within a reasonable limit rate when utilization goes down to encourage borrowing and vice versa. This makes the protocol vulnerable to interest rate manipulation economic attacks where attacker can borrow large sums to increase interest rates for users.

The documentation states that the protocol implements protection against such attacks when there is a surge in interest rates by increasing the protocol fee to 100% for 3 days. Thus, attackers cannot collect the fee from interest earned in liquidity pools to minimize the losses to attackers. This makes the attack economically infeasible for the attacker.

This protection is partially implemented in the `_accrue()` function in `cog_pair.vy`.

```
if dt > 86400:
    # if interest rate is increasing
    if (
        _accrue_info.interest_per_second
        > self.surge_info.last_interest_per_second
    ):
        # If daily change in interest rate is greater than Surge threshold, trigger surge breaker
        dr: uint64 = (
            _accrue_info.interest_per_second
            - self.surge_info.last_interest_per_second
        )
        if dr > PROTOCOL_SURGE_THRESHOLD:
            self.surge_info.last_elapsed_time = convert(
                block.timestamp, uint64
            )
            self.surge_info.last_interest_per_second = (
                _accrue_info.interest_per_second
            )
            # PoL Should accrue here, instead of to lenders, to discourage pid attacks as described in https://gauntlet.network/reports/pid
            self.protocol_fee = PROTOCOL_FEE_DIVISOR  # 100% Protocol Fee
    else:
        # Reset protocol fee elsewise
        self.protocol_fee = self.DEFAULT_PROTOCOL_FEE  # 10% Protocol Fee
  self.accrue_info = _accrue_info
```

Figure 3.4: Snippet for `_accrue()` in `cog_pair.vy`

The stated protection is not implemented in the `_accrue()` function. The increased protocol fee is not held up for 3 days.

Also, the protocol checks for surge at the most once per day. This makes the protocol vulnerable to interest rate manipulation attacks for one day.

Impact This makes protocol vulnerable to interest manipulation attacks stated above.

Recommendation

- The protocol should implement the stated protection.
- The protocol should check if surge happens more frequently.

Developer Response Developers acknowledged the issue. They’ve introduced a safeguard against interest rate spikes, rendering the attacks financially unfeasible within the reasonable limits they’ve set.

3.4.4 V-COG-VUL-004: Redeem returns wrong number of assets transferred

Severity	High	**Commit**	7ca2a6a
Type	Logic Error	**Status**	Fixed
File(s)	`src/cog_pair.vy`		
Location(s)	redeem()		

The protocol allows liquidity providers to redeem their shares to withdraw deposited asset tokens via the `redeem()` function. The `redeem()` function takes the the number of shares the users wishes to withdraw and returns the number of assets transferred to the withdrawer.

```
@external
def redeem(
    shares: uint256, receiver: address = msg.sender, owner: address = msg.sender
) -> uint256:
    """
    @param shares - The amount of shares to redeem
    @param receiver - The address of the receiver
    @param owner - The address of the owner

    @return - The amount of assets returned
    """
    self.efficient_accrue()
    assets_out: uint256 = self._convertToAssets(
        self._remove_asset(receiver, owner, shares)
    )
    log Withdraw(msg.sender, receiver, owner, assets_out, shares)

    return assets_out
```

Figure 3.5: `redeem()` in `cog_pair.vy`

The `redeem()` function performs following steps:

1. Accrues interest.
2. Calls the internal implementation `self._remove_asset()` to perform the required book keeping and transfer the tokens to withdrawer. It returns the number of assets transferred to user.
3. Erroneously, calls `self._convertToAssets()` to convert the returned quantity to assets.
4. Returns the `self._convertToAssets()` result.

Impact The `redeem()` function returns the inconsistent quantity returned by `self._convertToAssets()` which is not equal to the actual assets transferred and might lead to withdrawer operating under the assumption that larger number of assets returned than actually returned. This might lead to financial losses and loss of credibility for protocol.

Recommendation Remove the call to `self._convertToAssets()` in `redeem()`.

Developer Response Developers have acknowledged and fixed this issue.

3.4.5 V-COG-VUL-005: Protocol transfers in fewer funds in repay()

Severity	High	**Commit**	7ca2a6a
Type	Logic Error	**Status**	Fixed
File(s)	`cog_pair.vy`		
Location(s)	liquidate()		

The protocol allows users to deposit collateral and borrow loans against the deposited collateral. The users can then repay the loan by calling the `repay()` function which just accrues interest and calls to internal implementation in `_repay()`.

```
@internal
def _repay(to: address, payment: uint256) -> uint256:
    """
    @param to: The address to repay the tokens for
    @param payment: The amount of asset to repay, in tokens
    @return: The amount of tokens repaid in shares
    """
    temp_total_borrow: Rebase = Rebase(
        {
            elastic: 0,
            base: 0,
        }
    )
    amount: uint256 = 0

    temp_total_borrow, amount = self.sub(self.total_borrow, payment, True)
    self.total_borrow = temp_total_borrow

    self.user_borrow_part[to] = self.user_borrow_part[to] - payment
    total_share: uint128 = self.total_asset.elastic
    assert ERC20(asset).transferFrom(
        msg.sender, self, payment, default_return_value=True
    )  # dev: Transfer Failed

    self.total_asset.elastic = total_share + convert(amount, uint128)
    return amount
```

Figure 3.6: `repay()` in cog_pair.vy

The `repay()` function takes 2 arguments:

- `to`: The user whose loan is repaid
- `payment`: The shares of loans that are being repaid

The relevant steps in the function are:

1. Calculates the new `total_borrow` after repayment in `temp_total_borrow` by calling:

```
temp_total_borrow, amount = self.sub(self.total_borrow, payment, True)
```

   a) The `self.sub()` function takes in a `total: Rebase` struct and `shares` as arguments and returns new `Rebase` struct after reducing `shares` from `total`.
   b) Note: This indicates `payment` is in units of shares.

    c) The `self.sub()` function also returns `amount` which is in number of asset tokens equivalent to `payment`.

2. Update `self.total_borrow` with `temp_total_borrow`.
3. Reduce `self.user_borrow_part[to]` with `payment`.
4. Load `self.total_asset.elastic` in `total_share`.
5. Transfer in `payment` amount of asset tokens from `msg.sender` by calling `asset.transferFrom()`.

As noted above, `payment` is in units of shares but this is the quantity of tokens transferred in from borrower.

Impact As the number of shares is less than or equal to the number of tokens, the protocol transfers in less number of tokens than what is owed to the protocol by the borrower.

Recommendation Transfer in `amount` instead of `payment` tokens.

Developer Response Developers have acknowledged and fixed this issue.

3.4.6 V-COG-VUL-006: Protocol transfers in fewer funds in liquidate()

Severity	High	**Commit**	7ca2a6a
Type	Logic Error	**Status**	Fixed
File(s)	`cog_pair.vy`		
Location(s)	liquidate()		

The protocol allows users to buy the collateral deposited by the borrowers when the borrowers become insolvent by repaying the loans. The protocol implements this in `liquidate()` function.

Relevant snippets from `liquidate()` in `cog_pair`

```
@external
def liquidate(user: address, max_borrow_parts: uint256, to: address):
    """
    @param user The user to liquidate
    @param max_borrow_parts The parts to liquidate
    @param to The address to send the liquidated tokens to
    """
    exchange_rate: uint256 = 0
    updated: bool = False  # Never used
    updated, exchange_rate = self._update_exchange_rate()
    self.efficient_accrue()
```

```
 if not self._is_solvent(user, exchange_rate):
        available_borrow_part: uint256 = self.user_borrow_part[user]
        borrow_part: uint256 = min(max_borrow_parts, available_borrow_part)
        self.user_borrow_part[user] = available_borrow_part - borrow_part

        borrow_amount: uint256 = self.to_elastic(
            _total_borrow, borrow_part, False
        )
```

```
       all_collateral_share += collateral_share
        all_borrow_amount += borrow_amount
        all_borrow_part += borrow_part

                ...

    assert ERC20(collateral).transfer(
        to, all_collateral_share, default_return_value=True
    )  # dev: Transfer failed

    assert ERC20(asset).transferFrom(
        msg.sender, self, all_borrow_part, default_return_value=True
    )  # dev: Transfer failed

    self.total_asset.elastic = self.total_asset.elastic + convert(
        all_borrow_part, uint128
    )
```

The `liquidate()` function takes 3 arguments:

- `user`: The user that is to be liquidated

- `max_borrow_parts`: The shares of loan that liquidators want to liquidate
- `to`: The destination address for transferring the collateral

The relevant steps in the function are:

1. Check if the user is insolvent.
2. Calculate the shares borrowed by user and calculate the min of the borrowed parts and the `max_borrow_parts` and stores it in `borrow_part`.
3. Update `self.user_borrow_parts[user]` with `self.user_borrow_parts[user]-borrow_part` .
4. Calculate the `borrow_part` shares to elastic and stored in `borrow_amount`.
5. Then protocol then checks if user has asked to buy out the whole loan but does not have enough collateral, in which case the protocol marks all of the borrower's collateral to be transferred to liquidator. Among other things stores `borrow_part` in `all_borrow_part`.
6. Performs internal bookkeeping to reflect liquidation.
7. Transfers the calculated collateral to liquidator.
8. Transfers in `all_borrow_part` from the liquidator by calling in `asset.transferFrom()`.

`borrow_part` is in units of shares as indicated by the line below which is a call to `self.to_elastic()`, which takes in shares:

```
borrow_amount: uint256 = self.to_elastic(
            _total_borrow, borrow_part, False
        )
```

Therefore, `all_borrow_part` is in units of shares as well but this is the number of assets that are transferred in by the protocol.

Impact As the number of shares is less than or equal to the number of tokens, the protocol transfers in less number of tokens than what is owed to the protocol by the liquidator.

Recommendation Transfer in `borrow_amount` instead of `all_borrow_part` tokens.

Developer Response Developers have acknowledged and fixed this issue.

3.4.7 V-COG-VUL-007: Wrong amount returns as shares from withdraw

Severity	Medium	**Commit**	7ca2a6a
Type	Logic Error	**Status**	Fixed
File(s)	`src/cog_pair.vy`		
Location(s)	withdraw()		

The protocol allows liquidity providers to withdraw asset tokens using `withdraw()` and `redeem()` external functions. The `withdraw()` function takes as argument, the number of `assets` to withdraw.

```
@external
def withdraw(
    assets: uint256, receiver: address = msg.sender, owner: address = msg.sender
) -> uint256:
    """
    @param assets - The amount of assets to withdraw
    @param receiver - Reciever of the assets withdrawn
    @param owner - The owners whose assets should be withdrawn

    @return - The amount of shares burned
    """
    self.efficient_accrue()
    shares_to_withdraw: uint256 = self._convertToShares(assets)
    shares: uint256 = self._remove_asset(receiver, owner, shares_to_withdraw)
    log Withdraw(msg.sender, receiver, owner, assets, shares)

    return shares
```

Figure 3.7: `withdraw()` in `cog_pair.vy`

This function performs following steps:

1. Accrue interest.
2. Calculate the number of shares equivalent to number of `assets`.
3. Call internal implementation `self._remove_asset()`. This returns the number of asset tokens returned to the withdrawer and stores this in `shares`.
4. Return `shares`.

The documentation states that the `withdraw()` function should return the amount of shares that are redeemed equivalent to `assets`.

Impact

- The `withdraw()` function returns the amount of tokens withdrawn while, according to documentation, the function should return number of shares. Since the actual number of tokens will be either greater than or equal to equivalent number of shares, the function will report higher number of shares as redeemed than actually redeemed. This may cause a user to operate on inflated values of shares burned which may lead to financial losses for the user.

- The name of the variable `shares` is inconsistent with the quantity returned by `self._remove_assets()` (i.e. assets).

Recommendation

- Return `shares_to_withdraw`.
- Rename the variable `shares` to `assets`.

Developer Response Developers have acknowledged and fixed this issue.

3.4.8 V-COG-VUL-008: Comparison of shares and ERC20 tokens

Severity	Medium	**Commit**	7ca2a6a
Type	Logic Error	**Status**	Fixed
File(s)	`src/cog_pair.vy`		
Location(s)	previewWithdraw()		

The `cog_pair` contract provides a external function `previewWithdraw()` for users to get the number of shares that will be withdrawn from the protocol for a given number of assets.

```
@view
@external
def previewWithdraw(assets: uint256) -> uint256:
    """
    @param assets - The amount of assets to withdraw
    @return - The amount of shares worth withdrawn
    @notice - Will revert if you try to preview withdrawing more assets than
    available currently in the vault's balance
    """
    return min(self._convertToShares(assets), ERC20(asset).balanceOf(self))
```

Figure 3.8: `previewWithdraw()` in `cog_pair.vy`

The function calculates minimum of the following:

- The number of shares equivalent to number of assets requested to withdraw as given by `self._convertToShares(assets)`
- The balance of the `cog_pair` in the `asset` contract in case the pair does not have enough assets to successfully process the withdrawal request

This compares two inconsistent quantities, shares and number of assets.

Impact `ERC20(asset).balanceOf(self)` will be greater than or equal to the correct quantity of shares that will be withdrawn. Therefore the `min()` function might return `self._convertToShares()` and protocol might not have enough assets to fulfill the return request.

Recommendation The function should return

```
min(self._convertToShares(assets), self._convertToShares(ERC20(asset).balanceOf(self)
    ))
```

Developer Response Developers have acknowledged and fixed this issue.

3.4.9 V-COG-VUL-009: Possible overflow while calculating mean price

Severity	Low	**Commit**	7ca2a6a
Type	Logic Error	**Status**	Fixed
File(s)	`src/fuse_box.vy`		
Location(s)	get()		

The protocol uses oracles to get price feeds of asset tokens and collateral tokens. The protocol uses the `fuse_box` contract to aggregate price feeds from up to four (4) oracles. The `fuse_box` calculates the mean price of active oracles in the `get()` function.

This function first calculates the sum of all the active oracles in `total_price` and divides it by `active_oracles`.

Impact This way of calculating mean is prone to overflow if one of the token prices has very high absolute values.

Recommendation Store all prices in `uint256[4]` and calculate the mean at the end using:

$$SUM\_i(x\_i/count) + SUM\_i(x\_i\%count)/count\{\}\forall i \in [0, 4)$$

Developer Response Developers have acknowledged and fixed this issue.

```
@external
def get() -> (bool, uint256):
    """
    @return bool Whether or not the oracle updated
    @return uint256 the price of the asset
    """
    fuses: DataSource[4] = self.fuse_box

    total_price: uint256 = 0
    active_oracles: uint256 = 0
    updated: bool = False

    if fuses[0].active:
        updated_0: bool = False
        price: uint256 = 0
        (updated_0, price) = IOracle(fuses[0].oracle_address).get()
        updated = updated or updated_0
        total_price += price
        active_oracles += 1

    if fuses[1].active:
        updated_1: bool = False
        price: uint256 = 0
        (updated_1, price) = IOracle(fuses[1].oracle_address).get()
        updated = updated or updated_1
        total_price += price
        active_oracles += 1

    if fuses[2].active:
        updated_2: bool = False
        price: uint256 = 0
        (updated_2, price) = IOracle(fuses[2].oracle_address).get()
        updated = updated or updated_2
        total_price += price
        active_oracles += 1

    if fuses[3].active:
        updated_3: bool = False
        price: uint256 = 0
        (updated_3, price) = IOracle(fuses[3].oracle_address).get()
        updated = updated or updated_3
        total_price += price
        active_oracles += 1

    return (updated, (total_price / active_oracles))
```

Figure 3.9: `get()` in `fuse_box.vy`

3.4.10 V-COG-VUL-010: Consider using ‘mul_div’ in more locations

Severity	Low	**Commit**	7ca2a6a
Type	Logic Error	**Status**	Won’t Fix
File(s)	`src/cog_pair.vy`		
Location(s)	See description		

The number of bits required to hold the result of a multiplication is at least `max(n, m)` and most `(m+n)` where `m` and `n` are the number of bits required to hold the value of the two operands. When `(m+n) > 256` the result could overflow the `uint256` data type. When the multiplication is immediately followed by a division (i.e. statements of the form `a * b / c`), the final result may not overflow the `uint256` data type even though the intermediate result would.

The following locations have these operations that are prone to intermediate value overflow:

Impact Vyper protects from overflows by reverting the transaction. Hence, transactions may revert unnecessarily.

Recommendation Consider using `mul_div` function from `cog_pair.vy` or restricting the inputs to smaller data types such as `uint128` (if applicable).

Developer Response Developers acknowledged this issue, but they don’t intend fix the issue because they believe the issue can occur only on very high values but the variables in protocol are limited at maximum value of `uint128` therefore, the issue may not arise.

```
# src/cog_pair.vy:36
base: uint256 = (elastic * convert(total.base, uint256)) / convert(
    total.elastic, uint256
)

# src/cog_pair.vy:42
(base * convert(total.elastic, uint256))
    / convert(total.base, uint256)

# src/cog_pair.vy:63
elastic: uint256 = (base * convert(total.elastic, uint256)) / convert(
    total.base, uint256
)

# src/cog_pair.vy:70
(elastic * convert(total.base, uint256))
    / convert(total.elastic, uint256)

# src/cog_pair.vy:520
return shareAmount * all_share / convert(_total_asset.base, uint256)

# src/cog_pair.vy:543
return assetAmount * total_asset_base / all_share

# src/cog_pair.vy:748
convert(_total_borrow.elastic, uint256)
    * convert(_accrue_info.interest_per_second, uint256)
    * elapsed_time
    / 1000000000000000000

# src/cog_pair.vy:768
fee_amount * convert(_total_asset.base, uint256) / full_asset_amount

# src/cog_pair.vy:782
convert(_total_borrow.elastic, uint256)
  * UTILIZATION_PRECISION
  / full_asset_amount

# src/cog_pair.vy:789
(MINIMUM_TARGET_UTILIZATION - utilization)
  * FACTOR_PRECISION
  / MINIMUM_TARGET_UTILIZATION
```

Figure 3.10: Locations vulnerable to arithmetic overflow

```
# src/cog_pair.vy:797
convert(_accrue_info.interest_per_second, uint256)
  * INTEREST_ELASTICITY
  / scale,

# src/cog_pair.vy:816
convert(_accrue_info.interest_per_second, uint256)
  * scale
  / INTEREST_ELASTICITY,

# src/cog_pair.vy:903
fraction = (amount * convert(_total_asset.base, uint256)) / all_share

# src/cog_pair.vy:941
amount: uint256 = (share * all_share) / convert(_total_asset.base, uint256)

# src/cog_pair.vy:985
fee_amount: uint256 = (
  amount * self.BORROW_OPENING_FEE
) / BORROW_OPENING_FEE_PRECISION

# src/cog_pair.vy:1060
collateral_share
  * (EXCHANGE_RATE_PRECISION / COLLATERIZATION_RATE_PRECISION)

# src/cog_pair.vy:1223
(borrow_amount * LIQUIDATION_MULTIPLIER * exchange_rate)
  / (LIQUIDATION_MULTIPLIER_PRECISION * EXCHANGE_RATE_PRECISION)
```

Figure 3.11: Locations vulnerable to arithmetic overflow

3.4.11 V-COG-VUL-011: Subtracting values of different units

Severity	Low	**Commit**	7ca2a6a
Type	Logic Error	**Status**	Fixed
File(s)	`src/cog_pair.vy`		
Location(s)	totalAssets()		

The protocol allows users to act as liquidity providers by depositing `asset` tokens. Liquidity providers can earn interest on deposited assets. The protocol distributes collected interest among the liquidity providers via a mechanism of shares. Each liquidity provider is provided shares against the deposited asset tokens. The value of shares goes on increasing as the protocol collects interest. The protocol uses the `Rebase` struct to maintain the assets deposited and borrowed by users.

```
struct Rebase:
    elastic: uint128
    base: uint128
```

Figure 3.12: `Rebase` struct

The field `elastic` tracks the absolute number of asset tokens and the field `base` tracks the current shares minted by the pair. These two fields tracks quantities in different units.

The protocol provides a function `totalAssets()` to get the total assets available with the protocol.

```
def totalAssets() -> uint256:
    """
    @return - Returns the total amount of assets owned by the vault
    """
    total_elastic: uint256 = convert(self.total_asset.elastic, uint256)
    _total_borrow: Rebase = self.total_borrow
    # This could maybe revert in the case of bad debt, is that desired?
    total_interest: uint256 = convert(
        _total_borrow.elastic - _total_borrow.base, uint256
    )  # Interest is the difference between elastic and base, since they start at 1:1
    return total_interest + total_elastic
```

Figure 3.13: `totalAssets()` in `cog_pair.vy`

The function `totalAssets()` calculates `total_interest` by subtracting `_total_borrow.base` from `_total_borrow.elastic`. As noted above, `_total_borrow.base` is in units of shares while `_total_borrow.elastic` is in units of absolute number of asset tokens. This subtraction will return an inconsistent quantity.

Impact As `base` will be less than or equal to the `elastic`, this would lead to interest being calculated to be higher than the actual interest accumulated with the protocol. In turn this will inflate the total assets present within the protocol.

Recommendation Convert `_total_borrow.base` to elastic first and then subtract from `_total_borrow.elastic`.

Developer Response Developers have acknowledged and fixed this issue.

3.4.12 V-COG-VUL-012: Check if atleast one oracle is active in fuse_box

Severity	Warning	**Commit**	7ca2a6a
Type	Data Validation	**Status**	Fixed
File(s)	`src/fuse_box.vy`		
Location(s)	get()		

The protocol uses oracles to get price feeds of asset tokens and collateral tokens. The protocol uses the `fuse_box` contract to aggregate price feeds from up to four (4) oracles. The `fuse_box` calculates the mean price of active oracles in the `get()` function.

The protocol calculates mean price by adding the prices of all active oracles and then dividing the total by the number of active oracles.

In the initial state of the protocol, none of the protocol may be active so total number of active oracles might be zero.

Impact This will cause the `get()` function to revert due to a divide by zero error.

Recommendation The protocol must assert that there is at least one active oracle when the `cog_pair` is initialized.

Developer Response Developers have acknowledged and fixed this issue.

```
@external
def get() -> (bool, uint256):
    """
    @return bool Whether or not the oracle updated
    @return uint256 the price of the asset
    """
    fuses: DataSource[4] = self.fuse_box

    total_price: uint256 = 0
    active_oracles: uint256 = 0
    updated: bool = False

    if fuses[0].active:
        updated_0: bool = False
        price: uint256 = 0
        (updated_0, price) = IOracle(fuses[0].oracle_address).get()
        updated = updated or updated_0
        total_price += price
        active_oracles += 1

    if fuses[1].active:
        updated_1: bool = False
        price: uint256 = 0
        (updated_1, price) = IOracle(fuses[1].oracle_address).get()
        updated = updated or updated_1
        total_price += price
        active_oracles += 1

    if fuses[2].active:
        updated_2: bool = False
        price: uint256 = 0
        (updated_2, price) = IOracle(fuses[2].oracle_address).get()
        updated = updated or updated_2
        total_price += price
        active_oracles += 1

    if fuses[3].active:
        updated_3: bool = False
        price: uint256 = 0
        (updated_3, price) = IOracle(fuses[3].oracle_address).get()
        updated = updated or updated_3
        total_price += price
        active_oracles += 1

    return (updated, (total_price / active_oracles))
```

Figure 3.14: `get()` in `fuse_box.vy`

3.4.13 V-COG-VUL-013: Unnecessary memory copy

Severity	Warning	**Commit**	7ca2a6a
Type	Gas Optimization	**Status**	Won't Fix
File(s)	`src/cog_pair.vy`		
Location(s)	See description		

Copying of a storage struct to a memory struct is commonly observed pattern across the code base. For example, in function `_remove_asset()` the storage struct

```
@internal
def _remove_asset(to: address, owner: address, share: uint256) -> uint256:
    """
    @param to The address to remove asset for
    @param share The amount of asset to remove, in shares
    @return The amount of assets removed
    """
    if owner != msg.sender:
        assert (
            self.allowance[owner][msg.sender] >= share
        ), "Insufficient Allowance"
        self.allowance[owner][msg.sender] -= share

    _total_asset: Rebase = self.total_asset
    all_share: uint256 = convert(
        _total_asset.elastic + self.total_borrow.elastic, uint256
    )
    amount: uint256 = (share * all_share) / convert(_total_asset.base, uint256)
```

Figure 3.15: `_remove_asset()` in `cog_pair.vy`

In the function above, the storage struct `self.total_asset` is copied into memory struct `total_asset`.

We have observed that such copy triggers copying of whole struct into memory and vice versa. It is possible to directly use fields of storage structs instead of copying it into memory structs. This leads to wastage of gas due to memory expansion and unnecessary reads and write to and from memory.

In the cases listed below, there are no writes to the copy and there are no intervening writes to the original struct before all uses of the copy. Therefore, the copy is unnecessary and all uses can be replaced with the original struct reference.

Impact Gas is wasted while copying the structs into memory, copying structs back to storage and memory expansion. This is loss of funds for the protocol.

Recommendation Read/Write the fields directly from the storage structs instead of copying them to memory and write back to storage.

```
# cog_pair.vy:92
total: Rebase = _total # parameter passing already creates a copy

# cog_pair.vy:109
total: Rebase = _total # parameter passing already creates a copy

# cog_pair.vy:492
_total_borrow: Rebase = self.total_borrow

# cog_pair.vy:513
_total_asset: Rebase = self.total_asset

# cog_pair.vy:717
_accrue_info: AccrueInfo = self.accrue_info

# cog_pair.vy:730
_accrue_info: AccrueInfo = accrue_info

# cog_pair.vy:744
_total_asset: Rebase = self.total_asset

# cog_pair.vy:894
_total_asset: Rebase = self.total_asset

# cog_pair.vy:937
_total_asset: Rebase = self.total_asset

# cog_pair.vy:1003
_total_asset: Rebase = self.total_asset

# cog_pair.vy:1057
_total_borrow: Rebase = self.total_borrow

# cog_pair.vy:1211
_total_borrow: Rebase = self.total_borrow

# cog_pair.vy:1308
_accrue_info: AccrueInfo = self.accrue_info
```

Figure 3.16: Code locations where unnecessary memory copy is performed

Developer Response Developers acknowledged this issue but they determine they are okay with gas expenditure.

3.4.14 V-COG-VUL-014: Divide before multiply can give incorrect 0 result

Severity	Warning	**Commit**	7ca2a6a
Type	Maintainability	**Status**	Fixed
File(s)	`src/cog_pair.vy`		
Location(s)	_is_solvent()		

The protocol allows users to deposit collateral tokens and borrow asset tokens against the deposited collateral. During borrowing, the protocol checks if the borrower has deposited enough collateral to borrow. The protocol checks the solvency in `_is_solvent()` function.

```
@internal
def _is_solvent(user: address, exchange_rate: uint256) -> bool:
    """
    @param user: The user to check
    @param exchange_rate: The exchange rate to use
    @return: Whether the user is solvent
    """
    borrow_part: uint256 = self.user_borrow_part[user]
    if borrow_part == 0:
        return True
    collateral_share: uint256 = self.user_collateral_share[user]
    if collateral_share == 0:
        return False

    _total_borrow: Rebase = self.total_borrow
    collateral_amt: uint256 = (
        (
            collateral_share
            * (EXCHANGE_RATE_PRECISION / COLLATERIZATION_RATE_PRECISION)
        )
        * COLLATERIZATION_RATE
    )

    borrow_part = self.user_borrow_part[user]
    borrow_part = self.mul_div(
        (borrow_part * convert(_total_borrow.elastic, uint256)),
        exchange_rate,
        convert(_total_borrow.base, uint256),
        False,
    )
```

Figure 3.17: `_is_solvent()` in `cog_pair.vy`

While calculating `collateral_amt`

There is a divide before multiply, where `EXCHANGE_RATE_PRECISION` is divided `COLLATERIZATION_RATE_PRECISION` before multiplying it with `collateral_share`. The integer division would truncate the result to zero (0) when `EXCHANGE_RATE_PRECISION` is less than `COLLATERIZATION_RATE_PRECISION`, reducing the whole RHS expression to zero (0).

Impact In future updates of the code base these constants may change. If `EXCHANGE_RATE_PRECISION` is set to less than `COLLATERIZATION_RATE_PRECISION`, it will cause the integer division to truncate

```
collateral_amt: uint256 = (
        (
            collateral_share
            * (EXCHANGE_RATE_PRECISION / COLLATERIZATION_RATE_PRECISION)
        )
        * COLLATERIZATION_RATE
    )
```

Figure 3.18: snippet from `_is_solvent()` in `cog_pair.vy()`

to zero (0).

Recommendation

- Add a comment near definition of these constants about the constraints on the value.
- Assert that `EXCHANGE_RATE_PRECISION > COLLATERIZATION_RATE_PRECISION` at the beginning of the protocol.

Developer Response Developers acknowledged and fixed this issue by adding the recommended comment.

3.4.15 V-COG-VUL-015: _isPaused() function name is confusing

Severity	Warning	**Commit**	7ca2a6a
Type	Maintainability	**Status**	Fixed
File(s)	`src/cog_pair.vy`		
Location(s)	_isPaused()		

In the `cog_pair.vy` contract, the protocol checks if pair is paused using a boolean state variable, `self.paused`. Various external functions in the protocol check if the protocol is paused using an internal function `_isPaused()`.

```
@internal
def _isPaused():
    assert (not self.paused)
```

Figure 3.19: Definition of `_isPaused()` in `cog_pair.vy`

The function `_isPaused()` reverts if `self.paused` is set (i.e if the pair is paused). Else, the function returns normally if the protocol is not paused. Therefore, the name of the function is inconsistent with the implementation.

Also, the `assert` statement does not have a revert message.

Impact The inconsistent naming might lead to confusion as the code base grows.

Recommendation Rename the function to `_notPaused()`.

Add a revert message to the `assert` statement.

Developer Response Developers have acknowledged and fixed this issue.

3.4.16 V-COG-VUL-016: Unused variable or dead code

Severity	Warning	**Commit**	7ca2a6a
Type	Maintainability	**Status**	Fixed
File(s)	`src/cog_pair.vy`		
Location(s)	_add_asset()		

Function `_add_asset()` implements the steps for book keeping and transfer of asset tokens from user to the protocol.

```
total_asset_share: uint256 = convert(_total_asset.elastic, uint256)
```

Figure 3.20: cog_pair.vy:895

The implementation defines a variable `total_asset_share` which is not used anywhere in function.

Function `borrow()` implements borrowing of assets tokens from the liquidity pool.

```
accrue_info: AccrueInfo = self.accrue_info
```

Figure 3.21: cog_pair.vy:1169

The implementation defines a variable `accrue_info` that is not used anywhere in the function.

The `__init__()` function in `cog_pair` sets up various constants and storage variables.

During the initialization, the storage variable `self.protocol_fee` is assigned twice. It is first assigned the value `100000` and later it is assigned `self.DEFAULT_PROTOCOL_FEE`. Therefore, the first assignment is dead code.

Impact This leads to minor wastage of gas due to unused variable and dead code.

Recommendation Remove the dead code and unused variable.

Developer Response Developers have acknowledged and fixed this issue.

```
@external
def __init__(
    _asset: address,
    _collateral: address,
    _oracle: address,
    min_target_utilization: uint256,
    max_target_utilization: uint256,
    starting_interest_per_second: uint64,
    min_interest: uint64,
    max_interest: uint64,
    elasticity: uint256,
):
    assert (
        _collateral != 0x0000000000000000000000000000000000000000
    ), "Invalid Collateral"
    collateral = _collateral
    asset = _asset
    oracle = _oracle
    self.DEFAULT_PROTOCOL_FEE = 100000
    self.protocol_fee = 100000  # 10%
    MINIMUM_TARGET_UTILIZATION = min_target_utilization
    MAXIMUM_TARGET_UTILIZATION = max_target_utilization
    STARTING_INTEREST_PER_SECOND = starting_interest_per_second
    MINIMUM_INTEREST_PER_SECOND = min_interest
    MAXIMUM_INTEREST_PER_SECOND = max_interest
    INTEREST_ELASTICITY = elasticity
    self.protocol_fee = self.DEFAULT_PROTOCOL_FEE  # 10%
    self.BORROW_OPENING_FEE = 50
    factory = msg.sender
```

Figure 3.22: `__init__()` in `cog_pair.vy`

3.4.17 V-COG-VUL-017: Use of magic number

Severity	Warning	**Commit**	7ca2a6a
Type	Maintainability	**Status**	Fixed
File(s)	`src/cog_pair.vy`		
Location(s)	_accrue()		

The protocol uses various constants like `UTILIZATION_PRECISION`, `FACTOR_PRECISION` etc. In the following code locations, the protocol uses integer literals for constant values instead of defining and using constants.

```
interest_accrued = (
    convert(_total_borrow.elastic, uint256)
    * convert(_accrue_info.interest_per_second, uint256)
    * elapsed_time
    / 1000000000000000000
)  # 1e18, or the divisor for interest per second
```

Figure 3.23: `_accrue()` at cog_pair.vy:751

```
dt: uint64 = (
    convert(block.timestamp, uint64) - self.surge_info.last_elapsed_time
)
if dt > 86400:
```

Figure 3.24: `_accrue()` at cog_pair.vy:828

Impact If these constants are updated in future, this implementation is prone to missing out on updating the constant literals.

Recommendation Use the defined constants instead of constant literals.

Developer Response Developers have acknowledged and fixed this issue.

3.4.18 V-COG-VUL-018: Inconsistent or missing documentation

Severity	Info	**Commit**	7ca2a6a
Type	Maintainability	**Status**	Fixed
File(s)	See Description		
Location(s)	See description		

Documentation is incorrect at the following locations:

```
# src/cog_factory.vy:173
@dev Sets the status of a priviledged user
# does not describe the current function accurately

# src/cog_factory.vy:195
def change_fee_to(new_owner: address):
# the variable name 'new_owner' is misleading, perhaps 'new_recipient' is better

# src/cog_factory.vy:197
@dev Returns the address to which protocol fees are sent.
# does not describe the current function accurately (i.e. there is no return value)

# src/cog_pair.vy:383
PROTOCOL_FEE_DIVISOR: constant(uint256) = 1000000
# inconsistent naming: divisor used here, precision used for similar constants

# src/cog_pair.vy:655
shares: uint256 = self._remove_asset(receiver, owner, shares_to_withdraw)
# inconsistent: '_remove_asset' documentation states it returns asset value, but
    variable here is named 'shares'

# src/cog_pair.vy:978
def _borrow(amount: uint256, _from: address, to: address) -> uint256:
    """
    @param to: The address to send the borrowed tokens to
    @param amount: The amount of asset to borrow, in tokens
    @return: The amount of tokens borrowed
    """
# missing documentation for '_from' parameter

# src/cog_pair.vy:1153
def borrow(
    amount: uint256, _from: address = msg.sender, to: address = msg.sender
) -> uint256:
    """
    @param to The address to send the borrowed tokens to
    @param amount The amount of asset to borrow, in tokens
    @return The amount of tokens borrowed
    """
# missing documentation for '_from' parameter
```

Figure 3.25: Snippets where documentation is incorrect

Impact Makes the code more difficult to understand and maintain.

Recommendation Recommendations inline above.

Developer Response Developers have acknowledged and fixed this issue.