Vaults Solver Midas Audit

Côme du Crest

2025-10-12

Table of contents

Vaults Solver Midas Audit

This document presents the findings of a smart contract audit conducted by Côme du Crest.

For any inquiry contact me by email at come.ducrest@gmail.com or via telegram @ComeduCrest.

Scope

The scope includes the single `VaultsSolverMidas` contract within `0xhyperbeat/DNCoreWriter` as of commit `0xb0e409d`.

Fixes have been implemented and the contract has been reviewed as of commit `0xb8059fc`

Context

The goal of `VaultsSolverMidas` is to allow users to instantly redeem mTokens from Midas into a different supported withdrawal tokens. This is done through the `instantRedeem()` function that converts the input mToken amount to USD and the USD value to output token amount before transferring the tokens to the recipient.

The contract also has a `deposit()` function to deposit accepted tokens in exchange for mTokens. The conversion is similar to the `instantRedeem()` function in that it converts the deposited token amount to USD value and the USD value to mToken output amount.

The last user function is `zapVaults()` which allows to deposit an mToken and receive a different mToken with the same conversion logic as the two previous functions.

Both `instantRedeem()` and `zapVaults()` charge a fee while `deposit()` does not.

The intent is that the mTokens and other output tokens are deposited by the admin of the contract.

The oracles for converting tokens values to and from USD are the ones used by the underlying Midas vaults.

Status

The report has been sent to the core developer.

All issues have been fixed or acknowledged and no issue remain.

Legal Information And Disclaimer

1. This report is based solely on the information provided by Dunamis Labs Ltd (the “Company”), with the assumption that the information provided is authentic, accurate, complete, and not misleading as of the date of this report. The auditor has not conducted any independent enquiries, investigations or due diligence in respect of the Company, its business or its operations.
2. Changes to the information contained in the documents, repositories and any other materials referenced in this report might affect or change the analysis and conclusions presented. The auditor is not responsible for monitoring, nor will we be aware of, any future additions, modifications, or deletions to the audited code. As such, the auditor does not assume any responsibility to update any information, content or data contained in this report following the date of its publication.
3. This report does not address, nor should it be interpreted as addressing, any regulatory, tax or legal matters, including but not limited to: tax treatment, tax consequences, levies, duties, data privacy, data protection laws, issues relating to the licensing of information technology, intellectual property, money laundering and countering the financing of terrorism, or any other legal restrictions or prohibitions. The auditor disclaims any liability for omissions or errors in the findings or conclusions presented in this report.
4. The views expressed in this report are solely my views regarding the specific issues discussed within this report. This report is not intended to be exhaustive, nor should it be construed as an assurance, guarantee or warranty that the code is free from bugs, vulnerabilities, defects or deficiencies. Different use cases may carry different risks, and integration with third-party applications may introduce additional risks.
5. This report is provided for informational purposes only and should not be used as the basis for making investment or financial decisions. This report does not constitute investment research and should not be viewed as an invitation, recommendation, solicitation or offer to subscribe for or purchase any securities, investments, products or services. The auditor is not a financial advisor, and this report does not constitute financial or investment advice.
6. The statements in this report should be considered as a whole, and no individual statement should be extracted or referenced independently.
7. To the fullest extent permitted by applicable laws, the auditor disclaims any and all other liability, whether in contract, tort, or otherwise, that may arise from this report or the use thereof.

Issues

[Fixed][High] instantRedeem() does not charge a fee

Summary

The function `instantRedeem()` does not lower the amount of tokens sent to the caller by the computed fee value. As such, it does not actually charge a fee to the user.

Vulnerability Detail

The function `instantRedeem()` is supposed to charge a fee to the user. It computes the fee value `instantRedeemFee` from the `tokenAmount` amount of tokens corresponding to the user's given amount of mTokens. However it transfers the total `tokenAmount` assets to the user and did not take out a fee:

```
1     function instantRedeem(
2         address _recipient,
3         address _vaultToken,
4         address _withdrawalAsset,
5         uint256 _amount
6     )
7         external
8     {
9         if (!supportedWithdrawalAssetsPerVault[_vaultToken][_withdrawalAsset])
              {
10            revert VaultInstantWithdrawalMidas__WithdrawalAssetNotSupported(
                  _vaultToken, _withdrawalAsset);
11        }
12        (uint256 usdAmount, address redemptionVault) = _convertMTokenToUSD(
              _vaultToken, _amount);
13        uint256 tokenAmount = _convertUSDToToken(redemptionVault,
              _withdrawalAsset, usdAmount);
14        uint256 instantRedeemFee = _calcInstantRedeemFee(_vaultToken,
              tokenAmount);
15        uint256 adminFee = _calcAdminFee(instantRedeemFee);
16        if (adminFee > 0 && adminFeeRecipient != address(0)) {
17            _withdrawalAsset.safeTransfer(adminFeeRecipient, adminFee);
18        }
19        uint256 amountToTransfer = _amount - instantRedeemFee;
20        _vaultToken.safeTransferFrom(msg.sender, address(this), _amount);
21        _withdrawalAsset.safeTransfer(_recipient, tokenAmount);  // @audit no
              fee taken
22        emit InstantRedeem(_vaultToken, _withdrawalAsset, _amount,
              amountToTransfer, instantRedeemFee, adminFee);
23    }
```

Note that the value `amountToTransfer` is strictly used in the event and does not represent any-

thing as the `_amount` it is computed from is in mToken and `instantRedeemFee` is in output asset value.

Impact

No fee is actually charged to the user and the protocol realises a loss in this conversion as Midas actually charges a fee for such actions.

Code Snippets

https://github.com/0xhyperbeat/DNCoreWriter/blob/b0e409d6e53ebf17cf881f3a323acdcd60c47b85/src/VaultsSolverMidas.sol#L87-L109

https://github.com/0xhyperbeat/DNCoreWriter/blob/b0e409d6e53ebf17cf881f3a323acdcd60c47b85/tests/VaultsSolverMidas.t.sol#L150

Recommendation

The amount of withdrawal asset that should be sent to the user should be `tokenAmount - instantRedeemFee`.

Fix the tests by recomputing the amount expected to be received by the user instead of using a broad `assertGt()` to assert the received any amount of tokens. You can use `assertApproxEqAbs()` as an assert method (see `docs`).

Response

This has been fixed as of commit `0x3f6115a` by taking the fee in mToken and computing the USD value based on `_amount - instantRedeemFee`.

[Fixed][High] zapVaults transfers incorrect fee

Summary

The function `zapVaults()` compute a fee based on the input amount of tokens to swap but transfers the output tokens to the fee recipient instead.

Vulnerability Detail

In `zapVaults()`, a fee is computed based on the amount of input token. The `amountAfterFee` value is computed from the input amount token minus the `zapFee`. This amount after fee is used as the base for the calculation of the amount of tokens to output.

However, the admin fee tokens are transferred to the admin fee recipient using `_toVaultToken.safeTransfer()` meaning that the fees are taken in the wrong token:

```
1     function zapVaults(address _fromVaultToken, address _toVaultToken, uint256
          _amount) external {
2         ...
3         uint256 zapFee = _calcZapFee(_fromVaultToken, _toVaultToken, _amount);
4         uint256 adminFee = _calcAdminFee(zapFee);
5         if (adminFee > 0 && adminFeeRecipient != address(0)) {
6             _toVaultToken.safeTransfer(adminFeeRecipient, adminFee);
7         }
8         uint256 amountAfterFee = _amount - zapFee;
9         (uint256 fromUSDAmount,) = _convertMTokenToUSD(_fromVaultToken,
              amountAfterFee);
10        address depositVault = depositVaults[_toVaultToken];
11        if (depositVault == address(0)) {
12            revert VaultInstantWithdrawalMidas__DepositVaultNotSet(
                  _toVaultToken);
13        }
14        uint256 toMTokenAmount = _convertUSDToMToken(depositVault,
              fromUSDAmount);
15        _fromVaultToken.safeTransferFrom(msg.sender, address(this), _amount);
16        _toVaultToken.safeTransfer(msg.sender, toMTokenAmount);
17        emit ZapVaults(_fromVaultToken, _toVaultToken, _amount, toMTokenAmount,
               zapFee, adminFee);
18    }
```

Impact

If 10 mTokenA converts into 1.000 mTokenB and the admin fee percent is set to 10%, a fee of 1 mTokenA should be sent to the admin fee recipient. However, a fee of 1 mTokenB will be sent instead.

The fee sent to the admin fee recipient uses the incorrect token.

Code Snippets

https://github.com/0xhyperbeat/DNCoreWriter/blob/b0e409d6e53ebf17cf881f3a323acdcd60c47b85/src/VaultsSolverMidas.sol#L140

Recommendation

Transfer the correct token to the admin fee recipient (`_fromVaultToken`).

Response

This has been fixed as of commit `0x3f6115a` by transferring `_fromVaultToken` as admin fee.

[Fixed][Low] Admin fee setting documented in percent, used in bps

Summary

The admin fee setting is documented to be provided in percent but is actually used in bps. The documentation of `_calcAdminFee()`, `setAdminFee()`, and the variable name itself `adminFeePct` indicate using a percent calculation. However the `adminFeePct` value is divided by `1e4` which is a bps calculation.

Vulnerability Detail

The documentation states using a percent fee value but the calculation uses a bps value:

```
contract VaultsSolverMidas ... {
    ...
    uint256 public constant BASE = 1e4;
    uint256 public adminFeePct;

    /// @notice Calculates the admin fee from a given amount
    /// @dev Fee is calculated as a percentage of the amount using the global admin fee rate
    /// @param _amount Amount to calculate fee on
    /// @return fee The calculated admin fee
    function _calcAdminFee(uint256 _amount) internal view returns (uint256 fee) {
        fee = _amount.mulDiv(adminFeePct, BASE);
    }

    /// @notice Sets the global admin fee percentage
    /// @dev Only callable by ADMIN_ROLE. Fee is expressed in basis points relative to BASE (1e4)
    /// @param _adminFee Fee percentage in basis points
    function setAdminFee(uint256 _adminFee) external onlyRole(ADMIN_ROLE) {
        adminFeePct = _adminFee;
        emit AdminFeeSet(_adminFee);
    }

    ...
}
```

Impact

If the admin sets the fee expecting a percent value to be used they will receive less fee than expected.

Code Snippets

https://github.com/0xhyperbeat/DNCoreWriter/blob/b0e409d6e53ebf17cf881f3a323acdcd60c47b85/src/VaultsSolverMidas.sol#L253-L255

Recommendation

Make the documentation explicit about expecting a bps value instead of a percent value.

Response

The variable `adminFeePct` has been renamed `adminFeeBps` as of commit `0x3f6115a`. The notice of the natspec of `setAdminFee()` and `_calcAdminFee()` still refer to percents.

[Fixed][Low] Conversion using Midas oracles may be incorrect for stable coins

Summary

The `VaultsSolverMidas` attempts to mimic the conversion rates of Midas vaults by using the `getDataInBase18()` function from its data feed. However, Midas vaults ignore the value received from these feeds if the converted token is a stable coin and use a constant `STABLECOIN_RATE = 1e18` value instead.

If Midas vaults are set with data feeds for stable coins that do not return 1e18 there will be a discrepancy in between the conversion rate used by `VaultsSolverMidas` and the one used by Midas vaults.

Vulnerability Detail

The function `ManageableVault._getTokenRate()` used by Midas vaults to get the exchange rate of a token overwrites the answer from the data feed by `1e18` if the converted token is a stable coin:

```
    uint256 public constant STABLECOIN_RATE = 10**18;

    function _getTokenRate(address dataFeed, bool stable)
        internal
        view
        virtual
        returns (uint256)
    {
        // @dev if dataFeed returns rate, all peg checks passed
        uint256 rate = IDataFeed(dataFeed).getDataInBase18();

        if (stable) return STABLECOIN_RATE;

        return rate;
    }
```

This functions is used for example to convert the value of a token to USD in `DepositVault`:

```
    function _convertTokenToUsd(address tokenIn, uint256 amount)
        internal
        view
        virtual
        returns (uint256 amountInUsd, uint256 rate)
    {
        require(amount > 0, "DV: amount zero");

        TokenConfig storage tokenConfig = tokensConfig[tokenIn];

        rate = _getTokenRate(tokenConfig.dataFeed, tokenConfig.stable);
        require(rate > 0, "DV: rate zero");

```

```
14          amountInUsd = (amount * rate) / (10**18);
15      }
```

However, `VaultsSolverMidas._convertTokenToUSD()` directly uses the underlying `getDataInBase18()` without considering whether the token is a stable coin or not:

```
1       function _convertTokenToUSD(
2           address _vaultToken,
3           address _depositAsset,
4           uint256 _amount
5       )
6           internal
7           view
8           returns (uint256 usdAmount, address depositVault)
9       {
10          depositVault = depositVaults[_vaultToken];
11          if (depositVault == address(0)) {
12              revert VaultInstantWithdrawalMidas__DepositVaultNotSet(_vaultToken)
                    ;
13          }
14          IMidasDepositVault.TokenConfig memory tokenConfig = IMidasDepositVault(
                depositVault).tokensConfig(_depositAsset);
15          usdAmount = _amount.mulDiv(IMidasDataFeed(tokenConfig.dataFeed).
                getDataInBase18(), 10 ** 18);
16      }
```

Impact

If `VaultsSolverMidas` is used with stable coins, the conversion rate used may be different from what is used in the underlying Midas vaults.

Users can abuse this discrepancy for example by getting cheap mTokens for overpriced stable coins on `VaultsSolverMidas` and withdrawing stable coins on Midas vaults before repeating the operation.

Code Snippets

https://github.com/0xhyperbeat/DNCoreWriter/blob/b0e409d6e53ebf17cf881f3a323acdcd60c47b85/src/VaultsSolverMidas.sol#L223-L238

https://github.com/midas-apps/contracts/blob/ba8596751be53f4461349316b380c1e7868782e5/contracts/abstract/ManageableVault.sol#L632-L644

Recommendation

Use the value `tokenConfig.stable` from the config received from the underlying Midas vault to determine whether the data feed's rate should be overwritten with `1e18`.

Response

Fixed as of commit `0xb8059fc` by overwriting the Midas oracles responses with `1e18` if the coin is configured as stable.

[Ack][Info] Midas fee is ignored

Summary

The `deposit()` function that allows to deposit an asset and receive mTokens in exchange does not charge a fee. It is more beneficial for users to deposit through `VaultsSolverMidas` than to go through the official Midas `DepositVault` deposit process which charges its own fee.

This means that if the `VaultsSolverMidas` intends to convert the deposited assets into mToken through Midas, it will realize a loss for the protocol.

Similarly, the function `instantRedeem()` does not take into account Midas' fee into consideration. However, it charges its own fee which may be enough to cover for Midas' fee.

Vulnerability Detail

The function `VaultsSolverMidas.deposit()` strictly converts the deposit amount to USD and to mToken to send that to the receiver. It does not take into account any fee:

```
1     function deposit(address _vaultToken, address _depositAsset, address
          _receiver, uint256 _amount) external {
2         if (!supportedDepositAssetsPerVault[_vaultToken][_depositAsset]) {
3             revert VaultInstantWithdrawalMidas__DepositAssetNotSupported(
                  _vaultToken, _depositAsset);
4         }
5         (uint256 usdAmount, address depositVault) = _convertTokenToUSD(
              _vaultToken, _depositAsset, _amount);
6         uint256 mTokenAmount = _convertUSDToMToken(depositVault, usdAmount);
7         _depositAsset.safeTransferFrom(msg.sender, address(this), _amount);
8         _vaultToken.safeTransfer(_receiver, mTokenAmount);
9         emit Deposit(_vaultToken, _depositAsset, _receiver, _amount,
              mTokenAmount);
10    }
```

On the contrary, the Midas function `depositInstant()` charges a fee to the user:

```
1     function depositInstant(
2         address tokenIn,
3         uint256 amountToken,
4         uint256 minReceiveAmount,
5         bytes32 referrerId,
6         address recipient
7     )
8         external
9         whenFnNotPaused(_DEPOSIT_INSTANT_WITH_CUSTOM_RECIPIENT_SELECTOR)
10    {
11        ...
12        CalcAndValidateDepositResult memory result = _depositInstant(
13            tokenIn,
```

```
            amountToken,
            minReceiveAmount,
            recipient
        );
        ...
    }

    function _depositInstant(
        address tokenIn,
        uint256 amountToken,
        uint256 minReceiveAmount,
        address recipient
    ) internal virtual returns (CalcAndValidateDepositResult memory result) {
        address user = msg.sender;

        result = _calcAndValidateDeposit(user, tokenIn, amountToken, true);

        ...
        _instantTransferTokensToTokensReceiver(
            tokenIn,
            result.amountTokenWithoutFee,
            result.tokenDecimals
        );

        if (result.feeTokenAmount > 0)
            _tokenTransferFromUser(
                tokenIn,
                feeReceiver,
                result.feeTokenAmount,
                result.tokenDecimals
            );

        mToken.mint(recipient, result.mintAmount);

        ...
    }
```

The calculation of the output mToken is similar to what is done in `VaultsSolverMidas` with addition of a fee:

```
    function _calcAndValidateDeposit(
        address user,
        address tokenIn,
        uint256 amountToken,
        bool isInstant
    ) internal returns (CalcAndValidateDepositResult memory result) {
        ...
        (uint256 amountInUsd, uint256 tokenInUSDRate) = _convertTokenToUsd(
            tokenIn,
            amountToken
        );
        result.tokenAmountInUsd = amountInUsd;
        result.tokenInRate = tokenInUSDRate;
        address userCopy = user;
        ...
```

```
16
17          result.feeTokenAmount = _truncate(
18              _getFeeAmount(userCopy, tokenIn, amountToken, isInstant, 0),
19              result.tokenDecimals
20          );
21          result.amountTokenWithoutFee = amountToken - result.feeTokenAmount;
22
23          uint256 feeInUsd = (result.feeTokenAmount * result.tokenInRate) /
24              10**18;
25
26          (uint256 mTokenAmount, uint256 mTokenRate) = _convertUsdToMToken(
27              result.tokenAmountInUsd - feeInUsd
28          );
29          result.mintAmount = mTokenAmount;
30          result.tokenOutRate = mTokenRate;
31          ...
32      }
```

`instantRedeem()` charges a fee but it does not take into account Midas' fee calculation when computing the amount of withdrawal assets to send out:

```
1       function instantRedeem(
2           address _recipient,
3           address _vaultToken,
4           address _withdrawalAsset,
5           uint256 _amount
6       )
7           external
8       {
9           if (!supportedWithdrawalAssetsPerVault[_vaultToken][_withdrawalAsset])
                {
10              revert VaultInstantWithdrawalMidas__WithdrawalAssetNotSupported(
                    _vaultToken, _withdrawalAsset);
11          }
12          (uint256 usdAmount, address redemptionVault) = _convertMTokenToUSD(
                _vaultToken, _amount);  // @audit a midas fee is charged before that
                 in RedemptionVault
13          uint256 tokenAmount = _convertUSDToToken(redemptionVault,
                _withdrawalAsset, usdAmount);
14          uint256 instantRedeemFee = _calcInstantRedeemFee(_vaultToken,
                tokenAmount);
15          uint256 adminFee = _calcAdminFee(instantRedeemFee);
16          if (adminFee > 0 && adminFeeRecipient != address(0)) {
17              _withdrawalAsset.safeTransfer(adminFeeRecipient, adminFee);
18          }
19          uint256 amountToTransfer = _amount - instantRedeemFee;
20          _vaultToken.safeTransferFrom(msg.sender, address(this), _amount);
21          _withdrawalAsset.safeTransfer(_recipient, tokenAmount);
22          emit InstantRedeem(_vaultToken, _withdrawalAsset, _amount,
                amountToTransfer, instantRedeemFee, adminFee);
23      }
```

On Midas' `RedemptionVault` the first step of an instant redeem is to call `_calcAndValidateRedeem()`

```
    function _redeemInstant(
        address tokenOut,
        uint256 amountMTokenIn,
        uint256 minReceiveAmount,
        address recipient
    )
        internal
        virtual
        returns (
            CalcAndValidateRedeemResult memory calcResult,
            uint256 amountTokenOutWithoutFee
        )
    {
        address user = msg.sender;

        calcResult = _calcAndValidateRedeem(
            user,
            tokenOut,
            amountMTokenIn,
            true,
            false
        );
        ...
    }
```

The `_calcAndValidateRedeem()` function charges a fee diminishing the amount of mToken later converted into output asset:

```
    function _calcAndValidateRedeem(
        address user,
        address tokenOut,
        uint256 amountMTokenIn,
        bool isInstant,
        bool isFiat
    ) internal view returns (CalcAndValidateRedeemResult memory result) {
        require(amountMTokenIn > 0, "RV: invalid amount");

        if (!isFreeFromMinAmount[user]) {
            uint256 minRedeemAmount = isFiat ? minFiatRedeemAmount : minAmount;
            require(minRedeemAmount <= amountMTokenIn, "RV: amount < min");
        }

        result.feeAmount = _getFeeAmount(
            user,
            tokenOut,
            amountMTokenIn,
            isInstant,
            isFiat ? fiatAdditionalFee : 0
        );

        ...

        result.amountMTokenWithoutFee = amountMTokenIn - result.feeAmount;
    }
```

Impact

Users can exchange assets for mTokens at a better rate on `VaultsSolverMidas` than on Midas' contracts. If the owner of the solver wants to exchange to mTokens back to assets, they will have to pay a fee realizing an overall loss for the protocol.

The situation may be the same for redeeming mTokens into other assets.

This may be intended behaviour for the protocol as a means to balance out the supply of mTokens / other assets on the contract.

Code Snippets

https://github.com/midas-apps/contracts/blob/ba8596751be53f4461349316b380c1e7868782e5/contracts/DepositVault.sol

https://github.com/midas-apps/contracts/blob/ba8596751be53f4461349316b380c1e7868782e5/contracts/RedemptionVault.sol

https://github.com/0xhyperbeat/DNCoreWriter/blob/b0e409d6e53ebf17cf881f3a323acdcd60c47b85/src/VaultsSolverMidas.sol#L87-L126

Recommendation

Acknowledge that this is intended behaviour.

Alternatively, calculate the fee Midas would take on a deposit / withdraw and charge that same fee in `deposit()` and `instantRedeem()`.

Response

The protocol team acknowledged that this is intended behaviour.

[Ack][Info] Complete reliance on Midas oracles

Summary

The solver completely rely on oracles controlled by Midas for its conversion. If Midas decides to siphon all the funds of the contract, they can do that by updating their oracles to devalue the output tokens or overvalue the input tokens.

Vulnerability Detail

The function to convert the mToken amount to USD value queries the oracle of underlying Midas vault:

```
    function _convertMTokenToUSD(
        address _vaultToken,
        uint256 _amount
    )
        internal
        view
        returns (uint256 usdAmount, address redemptionVault)
    {
        redemptionVault = redemptionVaults[_vaultToken];
        if (redemptionVault == address(0)) {
            revert VaultInstantWithdrawalMidas__RedemptionVaultNotSet(
                _vaultToken);
        }
        address mTokenDataFeed = IMidasRedemptionVault(redemptionVault).
            mTokenDataFeed();
        usdAmount = IMidasDataFeed(mTokenDataFeed).getDataInBase18().mulDiv(
            _amount, 10 ** 18);
    }
```

The Midas oracle gets the answer by querying an aggregator round data:

```
    function getDataInBase18() external view returns (uint256 answer) {
        (, answer) = _getDataInBase18();
    }

    function _getDataInBase18()
        private
        view
        returns (uint80 roundId, uint256 answer)
    {
        uint8 decimals = aggregator.decimals();
        (uint80 _roundId, int256 _answer, , uint256 updatedAt, ) = aggregator
            .latestRoundData();
        require(_answer > 0, "DF: feed is deprecated");
        require(
            // solhint-disable-next-line not-rely-on-time
            block.timestamp - updatedAt <= healthyDiff &&
```

```
                _answer >= minExpectedAnswer &&
                _answer <= maxExpectedAnswer,
            "DF: feed is unhealthy"
        );
        roundId = _roundId;
        answer = uint256(_answer).convertToBase18(decimals);
    }
```

The used aggregator can be updated at any time by Midas admin:

```
    function changeAggregator(address _aggregator)
        external
        onlyRole(feedAdminRole(), msg.sender)
    {
        require(_aggregator != address(0), "DF: invalid address");

        aggregator = AggregatorV3Interface(_aggregator);
    }
```

Impact

Midas admins can set up a forged aggregator at any point in time to control the exchange rate used by `VaultsSolverMidas` and profit from that to siphon all the funds held by the contract.

Code Snippets

https://github.com/0xhyperbeat/DNCoreWriter/blob/b0e409d6e53ebf17cf881f3a323acdcd60c47b85/src/VaultsSolverMidas.sol#L173

https://github.com/midas-apps/contracts/blob/ba8596751be53f4461349316b380c1e7868782e5/contracts/feeds/DataFeed.sol#L74-L81

Recommendation

Acknowledge that Midas is trusted with the funds held by `VaultsSolverMidas`.

Response

The underlying Midas vaults will be owned by the protocol themselves so relying on them is not an issue.

Optimisations and miscellaneous

This part lists minor gas/code optimizations that shouldn't make the code less readable or improve overall readability. It also lists questions about unclear code segments.

Order of arguments in redeem and deposit do not match

The arguments used in the functions `instantRedeem()` and `deposit()` are similar but their orders do not match which may lead to errors when the functions are called.

`instantRedeem()` uses `(recipient, vaultToken, withdrawalAsset, amount)` while `deposit()` uses `(vaultToken, depositAsset, receiver, amount)`.

https://github.com/0xhyperbeat/DNCoreWriter/blob/b0e409d6e53ebf17cf881f3a323acdcd60c47b85/src/VaultsSolverMidas.sol#L117