Midas Audit

Côme du Crest

2024-08-09

Table of contents

Midas Audit

This document presents the finding of a smart contract audit conducted by Côme du Crest for Midas Protocol.

Scope

The scope includes all contracts within edDuck-Software/midas-contracts/tree/private-audit/contracts as of commit c235631 excluding `contracts/mocks` and `contracts/testers` directories expected to hold contracts necessary for testing.

Context

The main components of the protocol consists of a “DepositVault” and a “RedemptionVault” admin-controlled contracts. Users deposit tokens into the deposit vault and get minted proprietary “mTBILL”, “mBASIS”, or “eUSD” tokens. They can later redeem their tokens on the redemption vault to burn them for registered tokens, hopefully with a surplus.

Deposits and redemptions can be instantly applied with an additional fee and daily limit or delayed and validated by the vault admin. Tokens are withdrawn from the user at the time they register a deposit/withdrawal and the admin is responsible for reimbursing the user in case the deposit/withdrawal request is denied.

A “Metaspec” document was given alongside the code. It was deemed important and discrepancies in between the spec and the code are reported as issues.

Status

The report has been sent to the core developer.

The developers reviewed the report and implemented fixes in pull request 55. All issues have been fixes or rightfully acknowledged by the development team.

Issues

[Med] Updating fees in between deposit request and approval leads to inconsistency

Summary

When users initiate a deposit request on a vault, they immediately pay the associated fee. When the request is approved, the fee value is recomputed and the amount of MToken they receive for their deposit is lowered by the fee amount.

If fees are updated in between the request and its approval, the fee recipient will receive a certain amount of TokenIn as fee but the user will receive a value of MToken corresponding to a different fee.

Vulnerability Detail

The tokenIn transfers occur instantly during the call to `depositRequest()`. Fees are sent to `feeReceiver` and the remaining tokens to `tokensReceiver`:

```
    function depositRequest(
        address tokenIn,
        uint256 amountToken,
        bytes32 referrerId
    )
        external
        whenFnNotPaused(this.depositRequest.selector)
        onlyGreenlisted(msg.sender)
        onlyNotBlacklisted(msg.sender)
        onlyNotSanctioned(msg.sender)
        returns (uint256)
    {
        address user = msg.sender;

        address tokenInCopy = tokenIn;
        uint256 amountTokenCopy = amountToken;
        bytes32 referrerIdCopy = referrerId;

        (
            uint256 tokenAmountInUsd,
            uint256 feeAmount,
            uint256 amountTokenWithoutFee,
            ,
            ,
            uint256 tokenOutRate,
            uint256 tokenDecimals
        ) = _calcAndValidateDeposit(user, tokenInCopy, amountTokenCopy,
            false);
```

```
28
29          _tokenTransferFromUser(
30              tokenInCopy,
31              tokensReceiver,
32              amountTokenWithoutFee,
33              tokenDecimals
34          );
35
36          if (feeAmount > 0)
37              _tokenTransferFromUser(
38                  tokenInCopy,
39                  feeReceiver,
40                  feeAmount,
41                  tokenDecimals
42              );
43
44          ...
45      }
```

When the request is accepted, fees are computed once again and applied to the amount of MToken the user receives:

```
1       function _approveRequest(
2           uint256 requestId,
3           uint256 newOutRate,
4           bool isSafe
5       ) private {
6           Request memory request = mintRequests[requestId];
7
8           ...
9
10          uint256 feeUsdAmount = _getFeeAmount(
11              request.sender,
12              request.tokenIn,
13              request.depositedUsdAmount,
14              false,
15              0
16          );  // @audit applies the fees in the future, if fees changed
                in between request and approval, there is a discrepancy
17
18          uint256 amountMToken = ((request.depositedUsdAmount -
                feeUsdAmount) *
19              (10**18)) / newOutRate;
20
21          mToken.mint(request.sender, amountMToken);
22
23          ...
24      }
25
26      function _getFeeAmount(
27          address sender,
```

```
28          address token,
29          uint256 amount,
30          bool isInstant,
31          uint256 additionalFee
32      ) internal view returns (uint256) {
33          if (amount == 0) return 0;
34          if (waivedFeeRestriction[sender]) return 0;
35
36          uint256 feePercent;
37          if (additionalFee == 0) {
38              TokenConfig storage tokenConfig = tokensConfig[token];
39              feePercent = tokenConfig.fee;
40          } else {
41              feePercent = additionalFee;
42          }
43
44          if (isInstant) feePercent += instantFee;
45
46          if (feePercent > ONE_HUNDRED_PERCENT) feePercent =
               ONE_HUNDRED_PERCENT;
47
48          return (amount * feePercent) / ONE_HUNDRED_PERCENT;
49      }
```

If vault admin changed the value for `tokensConfig[token].fee` or `waivedFeeRestriction[sender]`, then the user could receive a different amount of MToken than requested.

Impact

Imagine there is a 20% fee, and one tokenIn equates to one MToken received. If a user initiate a deposit request with `100` tokenIn, `20` tokenIn will be sent to `feeReceiver` and `80` tokenIn will be sent to `tokensReceiver`.

If the vault admin updates the fee to lower them to 10% (or to waive fees for the user), the user will receive `90` MToken (or `100`) while `tokensReceive` received only `80` tokens to compensate for it.

On the other way around, if fees are increased (or user no longer has waived fees) the user will receive less MToken than anticipated and fees will be underpaid to `feeReceiver`.

Code Snippets

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/DepositVault.sol#L319-L330

Recommendation

Apply the fees at the time of the deposit request and store the MToken to be sent out in the request struct instead of `request.depositedUsdAmount`.

Response

This issue has been fixed in commit 8eb364e by calculating the fees only once at deposit request time and using a new variable `usdAmountWithoutFees` in `_approveRequest()`.

[Med] Vaults do not have special consideration towards stable coins

Summary

The specification states:

> For each token_in or token_out, the admin can add and update if the token is a stablecoin (as we have some peg checks)

However, this is not the case in the code.

Regarding deposit vaults, the specification states that:

> When the token_in is a stablecoin, the minting can only happen if the token_in is within its peg [0.997, 1.003] according to the chainlink oracle, and we apply 1 stablecoin = 1 USD for the exchange rate.

This is not enforced in the code. The code is able to use an oracle to fetch the price of a tokenIn and revert if the price is too high or low. However, it will use the price of the tokenIn and not enforce 1 stable coin = 1 USD.

Regarding redemption vaults, the specification states that:

> When the token_out is a stablecoin, the minting can only happen if the token_out is within its peg [0.997, 1.003] according to the chainlink oracle, and we apply 1 stablecoin = 1 USD for the exchange rate.

This is also not enforced in the code.

Vulnerability Detail

Adding a token to be used as tokenIn or tokenOut in a vault does not take into consideration whether the token is a stable coin:

```
    function addPaymentToken(
        address token,
        address dataFeed,
        uint256 tokenFee
    ) external onlyVaultAdmin {
        require(_paymentTokens.add(token), "MV: already added");
        _validateAddress(dataFeed, false);
        _validateFee(tokenFee, false);

        tokensConfig[token] = TokenConfig({
```

```
            dataFeed: dataFeed,
            fee: tokenFee,
            allowance: MAX_UINT
        });
        emit AddPaymentToken(token, dataFeed, tokenFee, msg.sender);
    }
```

The conversion rate of “1 USD = 1 stablecoin” is not enforced in token conversions:

```
contract DepositVault is ManageableVault, IDepositVault {

    ...

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

        TokenConfig storage tokenConfig = tokensConfig[tokenIn];

        rate = IDataFeed(tokenConfig.dataFeed).getDataInBase18();
        require(rate > 0, "DV: rate zero");

        amountInUsd = (amount * rate) / (10**18);  // @audit does not
            apply 1 USD = 1 stablecoin as specified in Metaspec (or no
            peg boundary enforced)
    }

    ...

}
```

```
contract RedemptionVault is ManageableVault, IRedemptionVault {

    ...

    function _convertUsdToToken(uint256 amountUsd, address tokenOut)
        internal
        view
        returns (uint256 amountToken, uint256 tokenRate)
    {
        require(amountUsd > 0, "RV: amount zero");

        TokenConfig storage tokenConfig = tokensConfig[tokenOut];

        tokenRate = IDataFeed(tokenConfig.dataFeed).getDataInBase18();
        require(tokenRate > 0, "RV: rate zero");

        amountToken = (amountUsd * (10**18)) / tokenRate;
```

```
18        }
19
20        ...
21
22    }
```

Impact

Behaviour of code differs from specification.

Code Snippets

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/abstract/ManageableVault.sol#L204-L219

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/DepositVault.sol#L415-L428

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L391-L404

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/feeds/DataFeed.sol#L86-L114

https://ludicrous-rate-748.notion.site/79bbe2aad23944e7af03bc122512edb7?v=5a4a00dce0844e288394a4f97c97ec04&p=54212964d2e644c3aee046120d8e52e1&pm=s

https://ludicrous-rate-748.notion.site/79bbe2aad23944e7af03bc122512edb7?v=5a4a00dce0844e288394a4f97c97ec04&p=ec554b68612a4e13a07cac74bc81f6fc&pm=s

https://ludicrous-rate-748.notion.site/79bbe2aad23944e7af03bc122512edb7?v=5a4a00dce0844e288394a4f97c97ec04&p=9c92573f7f984f89b1c8aa1144627df0&pm=s

Recommendation

Update the specification or add a field in the `TokenConfig` struct to specify whether a token is a stable coin and use a rate of 1 stable coin = 1 USD instead of the conversion rate from the oracle. You should still check that the oracle price is within the boundary `[0.997, 1.003]` as specified.

Response

This issue has been fixed in commit 72f642a by adding a field `stable` to `TokenConfig` and returning a constant rate of 1 stable coin = 1 USD when querying the conversion rate.

[Med] RedemptionVault does not pull tokens from wallet to pay out user

Summary

The specification states:

> Redemption Wallet: This is the wallet/address from which the contract can pull tokens from when a standard redemption request is processed.

However, the redemption wallet uses its own funds to pay out users during a redemption. Note that the specification conflicts with itself regarding that point when defining redemptions.

Vulnerability Detail

A redemption will use `_tokenTransferToUser()` when approving a redeem request which will use the contract's funds instead of an external wallet's funds:

```
    function _approveRequest(
        uint256 requestId,
        uint256 newMTokenRate,
        bool isSafe
    ) internal {

        ...

        if (request.tokenOut != MANUAL_FULLFILMENT_TOKEN) {
            ...

            _tokenTransferToUser(
                request.tokenOut,
                request.sender,
                amountTokenOutWithoutFee,
                tokenDecimals
            );
        }

        ...
    }

    function _tokenTransferToUser(
        address token,
        address to,
        uint256 amount,
        uint256 tokenDecimals
    ) internal {
        uint256 transferAmount = amount.convertFromBase18(tokenDecimals
            );
```

```
30
31          require(
32              amount == transferAmount.convertToBase18(tokenDecimals),
33              "MV: invalid rounding"
34          );
35
36          IERC20(token).safeTransfer(to, transferAmount);
37      }
```

Impact

Discrepancy in between specification and code

Code Snippets

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L268-L302

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/abstract/ManageableVault.sol#L418-L432

https://ludicrous-rate-748.notion.site/79bbe2aad23944e7af03bc122512edb7?v=5a4a00dce0844e288394a4f97c97ec04&p=a85a097a4a514f42b77b01d6857daf18&pm=s

Recommendation

Update the documentation or use an additional storage value for sender of tokens in redeems of `RedemptionVault`.

Response

Fixed in commit 6e79f0c by using a dedicated value `requestRedeemer` from which the tokens will be pulled during redeems.

[Low] No slippage parameters on vaults

Summary

There is no slippage parameter on deposit and withdraw requests on the vault. Users may send a deposit or withdraw transaction (instant or via request) and not know the trade rate that will be applied for their transaction.

Vulnerability Detail

Depositing tokens in the vault is a trade from tokenIn to MToken. Withdrawing tokens from the vault is a trade from MToken to tokenOut. The trading rate can be impacted by fees set by the vault admin, and by the oracle rates of tokenIn and MToken.

There is no input user value limiting the amount of tokens they receive as a result of the trade. Users accept whatever price is currently applied by the vault when their transactions are applied.

There is some control enforced by `variationTolerance` but this value can be updated at any point by the vault admin with no concerns to the user.

Impact

Users may receive less output token than anticipated when they signed their transaction if the state of the blockchain changes in between their signature and their transaction being applied.

Code Snippets

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/DepositVault.sol#L88-L92

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/DepositVault.sol#L149-L153

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L101-L107

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L160-L167

Recommendation

Impose a slippage parameter that user provides specifying how many MTokens / tokenOut they should receive as a result of a deposit / redeem.

Response

Slippage parameters have been added for instant deposits and redeems in commit e6d9e26. There is no slippage parameter for delayed deposits and redeems.

[Low] RedemptionVault does not validate tokenOutRate in redeem

Summary

When a redemption request is approved, the redemption vault does not validate the `tokenOutRate` value which may be outdated.

Vulnerability Detail

The function `_approveRequest()` responsible for approving a request checks that the `request.mTokenRate` is not too far from the current `newMTokenRate` value. It does not perform this check for `request.tokenOutRate` which may be outdated:

```
    function _approveRequest(
        uint256 requestId,
        uint256 newMTokenRate,
        bool isSafe
    ) internal {
        Request memory request = redeemRequests[requestId];

        _validateRequest(request.sender, request.status);

        if (isSafe) {
            _requireVariationTolerance(request.mTokenRate,
               newMTokenRate);
        }  // @audit no validation on request.tokenOutRate

        mToken.burn(address(this), request.amountMToken);

        if (request.tokenOut != MANUAL_FULLFILMENT_TOKEN) {
            uint256 tokenDecimals = _tokenDecimals(request.tokenOut);

            uint256 amountTokenOutWithoutFee = _truncate(
                (request.amountMToken * newMTokenRate) / request.
                   tokenOutRate,  // @audit may use outdated rate
                tokenDecimals
            );

            _tokenTransferToUser(
                request.tokenOut,
                request.sender,
                amountTokenOutWithoutFee,
                tokenDecimals
            );
        }

        request.status = RequestStatus.Processed;
```

```
33          request.mTokenRate = newMTokenRate;
34          redeemRequests[requestId] = request;
35      }
```

Impact

If the exchange rate of tokenOut varies in between the redeem request and its approval, the amount of token received may be unexpected.

Code Snippets

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L268-L302

Recommendation

This issue may be acknowledged if this is desired behaviour. Otherwise, verify that the tokenOutRate did not change too much using a provided value or oracle.

Response

This issue has been acknowledged.

[Info] IDepositVault documentation of depositInstant() misleading

Summary

The in-code documentation of `depositInstant()` states that `amountToken` is the "amount of `tokenIn` that will be taken from user", however the value that needs to be provided is scaled to 18 decimals and needs not to match the decimals used by `tokenIn`.

Vulnerability Detail$

The documentation is not explicit as to the expected value for `amountToken` as it states it is the amount of `tokenIn` that will be taken from user, however this value will be rescaled from `1e18` decimals to the `tokenIn` decimals before token from the user.

```
    /**
     * @notice depositing proccess with auto mint if
     * account fit daily limit and token allowance.
     * Transfers token from the user.
     * Transfers fee in tokenIn to feeReceiver.
     * Mints mToken to user.
     * @param tokenIn address of tokenIn
     * @param amountToken amount of `tokenIn` that will be taken from
        user  // @audit scaled to 1e18 decimals
     * @param referrerId referrer id
     */
    function depositInstant(
        address tokenIn,
        uint256 amountToken,
        bytes32 referrerId
    ) external;
```

Impact

Users reading the function's documentation may end up sending the wrong amount of token.

Code Snippets

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/interfaces/IDepositVault.sol#L120

Recommendation

Fix the documentation to specify the decimals scaling.

Response

The documentation has been updated in commit 1d5d65d

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.

Storage gap value uncommon size

The different upgradeable contracts use a `uint256[50] private __gap` value to provision for the possible added storage variables in future versions. It is a good idea, however it is more common and expected to provision for `50` total storage slot per contract and thus set the size of the gap to `50 - already_allocated_storage_slots` so that each contract takes exactly `50` slots. This make it easier to analyze the storage layout of the contracts once deployed.

For `DepositVault` for example that would be `uint256[47] private __gap`.

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/DepositVault.sol#L41

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L43

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L43

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/abstract/ManageableVault.sol#L128

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/access/WithMidasAccessControl.sol#L29

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/feeds/DataFeed.sol#L42

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/mTBILL/mTBILL.sol#L23

Response: this has been acknowledged.

Useless copying of values

In many places of the code, values are copied from calldata to memory seemingly for no reason. The new values are not modified and the copied value could have been used. Additionally, values are sometimes copied from called function to return values. This is a waste of gas and should be avoided.

```
    function _calcAndValidateDeposit(
         address user,
         address tokenIn,
         uint256 amountToken,
         bool isInstant
     )
         internal
         returns (
             uint256 tokenAmountInUsd,
             uint256 feeTokenAmount,
             uint256 amountTokenWithoutFee,
             uint256 mintAmount,
             uint256 tokenInRate,
             uint256 tokenOutRate,
             uint256 tokenDecimals
         )
     {
         require(amountToken > 0, "DV: invalid amount");

         tokenDecimals = _tokenDecimals(tokenIn);

         _requireTokenExists(tokenIn);

         (uint256 amountInUsd, uint256 tokenInUSDRate) =
            _convertTokenToUsd(
             tokenIn,
             amountToken
         );
         tokenAmountInUsd = amountInUsd;  // @audit useless
         tokenInRate = tokenInUSDRate;  // @audit useless
         address userCopy = user;  // @audit useless

         ...
     }
```

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/DepositVault.sol#L377-L383

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/DepositVault.sol#L132-L133

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/DepositVault.sol#L395-L399

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L119-L120

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L345

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L353

Response: acknowledged.

No validation on flat fees

The initializer of RedemptionVault validates the `fiatAdditionalFee` value but not the `fiatFlatFee` value which could be incorrect:

```
    function initialize(
        address _ac,
        address _mToken,
        address _tokensReceiver,
        address _feeReceiver,
        uint256 _instantFee,
        uint256 _instantDailyLimit,
        address _mTokenDataFeed,
        address _sanctionsList,
        uint256 _minAmount,
        uint256 _minFiatRedeemAmount,
        uint256 _variationTolerance,
        FiatRedeptionInitParams calldata _fiatRedemptionInitParams
    ) external initializer {
        __ManageableVault_init(
            _ac,
            _mToken,
            _tokensReceiver,
            _feeReceiver,
            _instantFee,
            _instantDailyLimit,
            _mTokenDataFeed,
            _sanctionsList,
            _variationTolerance,
            _minAmount
        );
        _validateFee(_fiatRedemptionInitParams.fiatAdditionalFee, false
            );

        minFiatRedeemAmount = _minFiatRedeemAmount;
        fiatAdditionalFee = _fiatRedemptionInitParams.fiatAdditionalFee
            ;
        fiatFlatFee = _fiatRedemptionInitParams.fiatFlatFee;  // @audit
            no validation on flat fee
    }
```

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L65-L96

Response: acknowledged.

ManageableVault.lastRequestId is not the last request ID

Since the value `lastRequestId` is updated after it is used, this value does not represent the ID of the last request, which is confusing. The value should be incremented before it is used.

```
    function depositRequest(
        address tokenIn,
        uint256 amountToken,
        bytes32 referrerId
    )
        external
        whenFnNotPaused(this.depositRequest.selector)
        onlyGreenlisted(msg.sender)
        onlyNotBlacklisted(msg.sender)
        onlyNotSanctioned(msg.sender)
        returns (uint256)
    {
        ...

        uint256 requestId = lastRequestId.current();
        lastRequestId.increment();

        mintRequests[requestId] = ...

        ...
    }
```

```
    function _redeemRequest(address tokenOut, uint256 amountMTokenIn)
        internal
        returns (uint256)
    {
        ...

        uint256 requestId = lastRequestId.current();
        lastRequestId.increment();

        redeemRequests[requestId] = ...

        ...
    }
```

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/DepositVault.sol#L192-L193

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/RedemptionVault.sol#L366-L367

Response: this has been fixed by updating the requestId before using it.

__ManageableVault_init() sets access control three times

The init function of `ManageableVault` calls chained init functions that set the value for `accessControl` three times. It would be cleaner to used unchained init functions the same way it is done with the `WithSanctionsList` contract.

```
    function __ManageableVault_init(
        address _ac,
        address _mToken,
        address _tokensReceiver,
        address _feeReceiver,
        uint256 _instantFee,
        uint256 _instantDailyLimit,
        address _mTokenDataFeed,
        address _sanctionsList,
        uint256 _variationTolerance,
        uint256 _minAmount
    ) internal onlyInitializing {
        ...

        mToken = IMTbill(_mToken);
        __Pausable_init(_ac);
        __Greenlistable_init(_ac);
        __Blacklistable_init(_ac);  // @audit sets access control three
            times, cleaner to use unchained
        __WithSanctionsList_init_unchained(_sanctionsList);

        ...
    }
```

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/abstract/ManageableVault.sol#L173-L176

Response: this has been fixed by using unchained init functions.

Access control roles are split across contracts

Most access control roles can be found in `MidasAccessControlRoles`, but some of them are split across other contracts such as `eUSD`, `EUsdMidasAccessControlRoles`, or `mBASIS`. I would say some separation makes sense like the `GREENLIST_TOGGLER_ROLE` defined in `Greenlistable` but it is overall confusing to have a roles contract and roles split in other contracts as well.

I would recommend deciding on putting all roles in a single contract with a definition of their rights or splitting them in their responsible contracts for ease of understanding.

Response: acknowledged.

Deprecated use of `_setupRole()`

The function `AccessControlUpgradeable._setupRole()` used by `MidasAccessControl` is deprecated and should be replaced with `_grantRole()`.

https://github.com/RedDuck-Software/midas-contracts/blob/c235631bb83317b688b1d146aea6f25a7a0ecca1/contracts/access/MidasAccessControl.sol#L70-L85

Response: `_setupRole()` has been replaced with `_grantRole()`.

Div before mul loses precision

In `CustomAggregatorV3CompatibleFeed._getDeviation()` there is an unnecessary division before a multiplication which lowers precision. The return value is `deviation * 100`, it would increase precision if instead we defined `one = int256(100 * 10**decimals())`.

```
1     function _getDeviation(int256 _lastPrice, int256 _newPrice)
2         internal
3         pure
4         returns (uint256)
5     {
6         if (_newPrice == 0) return 100 * 10**decimals();
7         int256 one = int256(10**decimals());
8         int256 priceDif = _newPrice - _lastPrice;
9         int256 deviation = (priceDif * one) / _lastPrice;
10        deviation = deviation < 0 ? deviation * -1 : deviation;
11        return uint256(deviation * 100);  // @audit div before mul
              loses precision
12    }
```

Response: this has been fixed by doing the multiplication before the division.