BLOCKSEC

Security Audit Report for Lista Dao Contracts

Date: August 06, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Lista
Target	Lista Dao Contracts

Version History

Version	Date	Description
1.0	August 06, 2024	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-08-06 05:44:57+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository of Lista Dao Contracts[1] of Lista. Note that, we did **NOT** audit all the modules in the repository. The modules covered by this audit report include `lista-dao-contracts` folder contract only. Specifically, the files covered in this audit include:

```
Interaction.sol
DynamicDutyCalculator.sol
IDao.sol
IDynamicDutyCalculator.sol
FixedMath0x.sol
```

Listing 1.1: Audit Scope for this Report

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Lista Dao Contracts	Version 1	b5770a310859608f177afa392a0686020c277210
	Version 2	236e1065815ab55e0de070c5256fda8865cad118

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does

[1] https://github.com/lista-dao/lista-dao-contracts/releases/tag/amo-audit-1.1

not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Permission management
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact

- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact	*High*	High	Medium
	Low	Medium	Low
		High	*Low*
		Likelihood	

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we found **six** potential security issues. Besides, we have **four** recommendations and **one** note.

- High Risk: 0
- Medium Risk: 3
- Low Risk: 3
- Recommendation: 4
- Note: 1

ID	Severity	Description	Category	Status
1	Low	Lack of rate update in function `withdraw()`	DeFi Security	Fixed
2	Medium	Lack of duty range check in function `calculateDuty()`	DeFi Security	Fixed
3	Medium	Incorrect calculations in function `estimatedLiquidationPriceHAY()`	DeFi Security	Confirmed
4	Medium	Incorrect duty returned in function `calculateDuty()`	DeFi Security	Confirmed
5	Low	Incorrect price used in function `borrow()`	DeFi Security	Fixed
6	Low	Lack of rate update in function `setCollateralParams()`	DeFi Security	Confirmed
7	-	Redundant check in function `setCollateralParms()`	Recommendation	Fixed
8	-	Comment mismatch with code logic	Recommendation	Fixed
9	-	Potential precision loss in function `liquidationPriceForDebt()`	Recommendation	Confirmed
10	-	Timely update role in function `file()`	Recommendation	Fixed
11	-	Potential centralization risk	Note	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Lack of rate update in function `withdraw()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `withdraw()` allows users to withdraw collateral from their vaults if the collateralization ratio is sufficient. However, the rate of the corresponding collateral, which changes over time, is not updated. The similar issue also exists in function `startAuction()`.

```
function withdraw(
    address participant,
    address token,
    uint256 dink
) external nonReentrant returns (uint256) {
    CollateralType memory collateralType = collaterals[token];
    _checkIsLive(collateralType.live);
    if (helioProviders[token] != address(0)) {
        require(
            msg.sender == helioProviders[token],
            "Interaction/Only helio provider can call this function for this token"
        );
    } else {
        require(
            msg.sender == participant,
            "Interaction/Caller must be the same address as participant"
        );
    }


    uint256 unlocked = free(token, participant);
    if (unlocked < dink) {
        int256 diff = int256(dink) - int256(unlocked);
        vat.frob(collateralType.ilk, participant, participant, participant, - diff, 0);
    }
    // move the dink amount of collateral from participant to the current contract
    vat.flux(collateralType.ilk, participant, address(this), dink);
    // Collateral is actually transferred back to user inside ‘exit‘ operation.
    // See GemJoin.exit()
    collateralType.gem.exit(msg.sender, dink);
    deposits[token] -= dink;


    emit Withdraw(participant, dink);
    return dink;
}
```

Listing 2.1: Interaction.sol

Impact The user can withdraw more collateral than expected.

Suggestion Invoke the function `drip()` to refresh the rate before validating the withdrawal request.

Feedback from the project Our bot will invoke the function `drip()` every two hours.

2.1.2 Lack of duty range check in function `calculateDuty()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description According to the protocol design, when the stablecoin price is less than or equal to the `minPrice`, the stablecoin interest rate should reach the maximum rate. However, the function `calculateDuty()` lacks the necessary checks to ensure this and directly uses the result generated from the current interest rate formula. This can cause the interest rate to exceed the maxDuty (maximum interest rate) before the stablecoin price falls below the minPrice. Based on the parameter settings used in the test file, the current protocol's stablecoin interest rate exceeds the maxDuty of 200% before the price falls to 0.9.

```solidity
function calculateDuty(address _collateral, uint256 _currentDuty, bool _updateLastPrice) public onlyRole(INTERACTION) returns (uint256 duty) {
    Ilk storage ilk = ilks[_collateral];
    if (!ilk.enabled) {
        return _currentDuty; // if collateral not enabled for dynamic interest rate mechanism, return current duty
    }
    uint256 price = oracle.peek(lisUSD);


    // return max duty if price is too low
    if (price <= minPrice) {
        if (_updateLastPrice) {
            ilk.lastPrice = price;
        }
        return maxDuty;
    }


    // return min duty if price is too high
    if (price >= maxPrice) {
        if (_updateLastPrice) {
            ilk.lastPrice = price;
        }
        return minDuty;
    }


    // return current duty if lastPrice - 0.002 <= price <= lastPrice + 0.002
    if (price <= ilk.lastPrice + priceDeviation && price >= ilk.lastPrice - priceDeviation) {
        return _currentDuty;
    }


    if (_updateLastPrice) {
        ilk.lastPrice = price;
    }


    uint256 rate = calculateRate(price, ilk.beta, ilk.rate0);
    duty = rate + 1e27;
}
```

Listing 2.2: DynamicDutyCalculator.sol

Impact Users will pay more interest than expected.

Suggestion Add relevant check in function calculateDuty().

Feedback from the project Lista team identified and fixed this issue before it was reported.

2.1.3 Incorrect calculations in function estimatedLiquidationPriceHAY()

Severity Medium

Status Confirmed

Introduced by Version 1

Description In the function estimatedLiquidationPriceHAY(), the current user's debt token amount is calculated using uint256 backedDebt = FullMath.mulDiv(art, rate, 10 ** 36). However, since the decimal of rate is 27, the denominator should be 10 ** 27 instead of 10 ** 36 to ensure that backedDebt has the same decimal as the debt token.

```
function estimatedLiquidationPriceHAY(address token, address usr, int256 amount) external view
     returns (uint256) {
    CollateralType memory collateralType = collaterals[token];
    _checkIsLive(collateralType.live);


    (uint256 ink, uint256 art) = vat.urns(collateralType.ilk, usr);
    require(amount >= - (int256(art)), "Cannot withdraw more than current amount");
    (, uint256 rate,,,) = vat.ilks(collateralType.ilk);
    (,uint256 mat) = spotter.ilks(collateralType.ilk);
    uint256 backedDebt = FullMath.mulDiv(art, rate, 10 ** 36);
    if (amount < 0) {
        backedDebt = uint256(int256(backedDebt) + amount);
    } else {
        backedDebt += uint256(amount);
    }
    return FullMath.mulDiv(backedDebt, mat, ink) / 10 ** 9;
}
```

Listing 2.3: Interaction.sol

Impact The function estimatedLiquidationPriceHAY() returns the incorrect value.

Suggestion Replace 10 ** 36 with 10 ** 27.

Feedback from the project Acknowledged. This issue will be fixed in our next version. Function estimatedLiquidationPriceHAY() is not being used by our Backend and Frontend.

2.1.4 Incorrect duty returned in function calculateDuty()

Severity Medium

Status Confirmed

Introduced by Version 1

Description The protocol is intended to dynamically adjust the interest rates for various collaterals based on the price of lisUSD. However, when invoking the function setCollateralParams() to configure corresponding variables for collaterals, lastPrice is not updated (default value is 0). In this case, the if statement in function calculateDuty() executed as false (line 152) when determining if the lisUSD price is within the range of lastPrice and priceDeviation (lastPrice +/- priceDeviation). This leads to the result that should return _currentDuty returning another value instead.

```
function calculateDuty(address _collateral, uint256 _currentDuty, bool _updateLastPrice) public onlyRole(INTERACTION) returns (uint256 duty) {
    Ilk storage ilk = ilks[_collateral];
    if (!ilk.enabled) {
        return _currentDuty; // if collateral not enabled for dynamic interest rate mechanism, return current duty
    }
    uint256 price = oracle.peek(lisUSD);


    // return max duty if price is too low
    if (price <= minPrice) {
        if (_updateLastPrice) {
            ilk.lastPrice = price;
        }
        return maxDuty;
    }


    // return min duty if price is too high
    if (price >= maxPrice) {
        if (_updateLastPrice) {
            ilk.lastPrice = price;
        }
        return minDuty;
    }


    // return current duty if lastPrice - 0.002 <= price <= lastPrice + 0.002
    if (price <= ilk.lastPrice + priceDeviation && price >= ilk.lastPrice - priceDeviation) {
        return _currentDuty;
    }


    if (_updateLastPrice) {
        ilk.lastPrice = price;
    }


    uint256 rate = calculateRate(price, ilk.beta, ilk.rate0);
    duty = rate + 1e27;
}
```

Listing 2.4: DynamicDutyCalculator.sol

```
function setCollateralParams(address collateral, uint256 beta, uint256 rate0, bool enabled) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(collateral != address(0), "AggMonetaryPolicy/invalid-address");
    require(beta > 3e5 && beta < 1e8, "AggMonetaryPolicy/invalid-beta");
    require(rate0 >= 0, "AggMonetaryPolicy/invalid-rate0");


    ilks[collateral].beta = beta;
```

```
        ilks[collateral].rate0 = rate0;
        ilks[collateral].enabled = enabled;


        emit CollateralParamsUpdated(collateral, beta, rate0, enabled);
    }
```

Listing 2.5: DynamicDutyCalculator.sol

Impact Incorrect interest rate will be updated.

Suggestion Retrieve and update the `lastPrice` in function `setCollateralParams()`.

Feedback from the project Yeah this is expected behaviour. We don't set `lastPrice` in `setCollateralParams()` in order to trigger the dynamic duty calculation when we go launch.

2.1.5 Incorrect price used in function `borrow()`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `Interaction` contract, users can borrow `lisUSD` through the function `borrow()`. The function `frob()` determines whether users can borrow based on the collateral price recorded in the contract. However, the collateral price recorded is not the latest. Specifically, anyone can invoke the function `poke()` to update with the latest price returned by the oracle, and the price used in the function `borrow()` is the one updated by function `poke()`. If `poke()` is not invoked for a long time, the price used during `borrow()` might significantly differ from the real market price, which is incorrect. The same issue also exists in the functions `payback()` and `withdraw()`.

```
    function borrow(address token, uint256 hayAmount) external notInBlacklisted(token)
         nonReentrant returns (uint256) {
        CollateralType memory collateralType = collaterals[token];
        require(collateralType.live == 1, "Interaction/inactive-collateral");


        drip(token);
        dropRewards(token, msg.sender);


        (, uint256 rate, , ,) = vat.ilks(collateralType.ilk);
        int256 dart = int256(hayAmount * RAY / rate);
        require(dart >= 0, "Interaction/too-much-requested");


        if (uint256(dart) * rate < hayAmount * RAY) {
            dart += 1; //ceiling
        }


        vat.frob(collateralType.ilk, msg.sender, msg.sender, msg.sender, 0, dart);
    vat.move(msg.sender, address(this), hayAmount * RAY);
    hayJoin.exit(msg.sender, hayAmount);


    (uint256 ink, uint256 art) = vat.urns(collateralType.ilk, msg.sender);
    uint256 liqPrice = liquidationPriceForDebt(collateralType.ilk, ink, art);
    emit Borrow(msg.sender, token, ink, hayAmount, liqPrice);
    return uint256(dart);
}
```

Listing 2.6: Interaction.sol

```
function frob(bytes32 i, address u, address v, address w, int dink, int dart) external auth {
  // system is live
  require(live == 1, "Vat/not-live");


  Urn memory urn = urns[i][u];
  Ilk memory ilk = ilks[i];
  // ilk has been initialised
  require(ilk.rate != 0, "Vat/ilk-not-init");


  urn.ink = _add(urn.ink, dink);
  urn.art = _add(urn.art, dart);
  ilk.Art = _add(ilk.Art, dart);


  int dtab = _mul(ilk.rate, dart);
  uint tab = _mul(ilk.rate, urn.art);
  debt     = _add(debt, dtab);


  // either debt has decreased, or debt ceilings are not exceeded
  require(either(dart <= 0, both(_mul(ilk.Art, ilk.rate) <= ilk.line, debt <= Line)), "Vat/ceiling-exceeded");
  // urn is either less risky than before, or it is safe
  require(either(both(dart <= 0, dink >= 0), tab <= _mul(urn.ink, ilk.spot)), "Vat/not-safe");


  // urn is either more safe, or the owner consents
  require(either(both(dart <= 0, dink >= 0), wish(u, msg.sender)), "Vat/not-allowed-u");
  // collateral src consents
  require(either(dink <= 0, wish(v, msg.sender)), "Vat/not-allowed-v");
  // debt dst consents
  require(either(dart >= 0, wish(w, msg.sender)), "Vat/not-allowed-w");


  // urn has no debt, or a non-dusty amount
  require(either(urn.art == 0, tab >= ilk.dust), "Vat/dust");


  gem[i][v] = _sub(gem[i][v], dink);
```

```
    hay[w]   = _add(hay[w],  dtab);


    urns[i][u] = urn;
    ilks[i]    = ilk;
}
```

Listing 2.7: vat.sol

```
function poke(address token) public {
    CollateralType memory collateralType = collaterals[token];
    _checkIsLive(collateralType.live);


    spotter.poke(collateralType.ilk);
}
```

Listing 2.8: Interaction.sol

```
function poke(bytes32 ilk) external {
    (bytes32 val, bool has) = ilks[ilk].pip.peek();
    uint256 spot = has ? rdiv(rdiv(mul(uint(val), 10 ** 9), par), ilks[ilk].mat) : 0;
    vat.file(ilk, "spot", spot);
    emit Poke(ilk, val, spot);
}
```

Listing 2.9: spot.sol

```
function file(bytes32 ilk, bytes32 what, uint data) external auth {
    require(live == 1, "Vat/not-live");
    if (what == "spot") ilks[ilk].spot = data;
    else if (what == "line") ilks[ilk].line = data;
    else if (what == "dust") ilks[ilk].dust = data;
    else revert("Vat/file-unrecognized-param");
}
```

Listing 2.10: vat.sol

```
function payback(address token, uint256 hayAmount) external nonReentrant returns (int256) {
    CollateralType memory collateralType = collaterals[token];
    // _checkIsLive(collateralType.live); Checking in the ‘drip‘ function


    dropRewards(token, msg.sender);
    drip(token);
    (,uint256 rate,,,) = vat.ilks(collateralType.ilk);
    (,uint256 art) = vat.urns(collateralType.ilk, msg.sender);


    int256 dart;
    uint256 realAmount = hayAmount;


    uint256 debt = rate * art;
    if (realAmount * RAY >= debt) { // Close CDP
        dart = int(art);
        realAmount = debt / RAY;
        realAmount = realAmount * RAY == debt ? realAmount : realAmount + 1;
    } else { // Less/Greater than dust
        dart = int256(FullMath.mulDiv(realAmount, RAY, rate));
    }


    IERC20Upgradeable(hay).safeTransferFrom(msg.sender, address(this), realAmount);
    hayJoin.join(msg.sender, realAmount);


    require(dart >= 0, "Interaction/too-much-requested");


    vat.frob(collateralType.ilk, msg.sender, msg.sender, msg.sender, 0, - dart);


    (uint256 ink, uint256 userDebt) = vat.urns(collateralType.ilk, msg.sender);
    uint256 liqPrice = liquidationPriceForDebt(collateralType.ilk, ink, userDebt);


    emit Payback(msg.sender, token, realAmount, userDebt, liqPrice);
    return dart;
}
```

Listing 2.11: Interaction.sol

```
function withdraw(
    address participant,
    address token,
    uint256 dink
) external nonReentrant returns (uint256) {
    CollateralType memory collateralType = collaterals[token];
    _checkIsLive(collateralType.live);
    if (helioProviders[token] != address(0)) {
        require(
            msg.sender == helioProviders[token],
            "Interaction/Only helio provider can call this function for this token"
        );
    } else {
        require(
            msg.sender == participant,
            "Interaction/Caller must be the same address as participant"
        );
    }


    uint256 unlocked = free(token, participant);
    if (unlocked < dink) {
        int256 diff = int256(dink) - int256(unlocked);
        vat.frob(collateralType.ilk, participant, participant, participant, - diff, 0);
    }
    // move the dink amount of collateral from participant to the current contract
    vat.flux(collateralType.ilk, participant, address(this), dink);
    // Collateral is actually transferred back to user inside ‘exit‘ operation.
    // See GemJoin.exit()
    collateralType.gem.exit(msg.sender, dink);
    deposits[token] -= dink;


    emit Withdraw(participant, dink);
    return dink;
}
```

Listing 2.12: Interaction.sol

Impact The collateral price used during borrowing may significantly differ from the actual market price.

Suggestion Promptly invoke the function `poke()`, or retrieve the latest collateral price from the oracle at the time of borrowing.

Feedback from the project Our bot will invoke the function `poke()` every two hours.

2.1.6 Lack of rate update in function `setCollateralParams()`

Severity Low

Status Confirmed

Introduced by `Version 1`

Description Function `setCollateralParams()` allows the privileged role to update the parameters of specific collateral for the calculation of dynamic interest `duty` (i.e., per-second stability fee). However, when these parameters change, the function `drip()` is not invoked immediately. In this case, the accumulated stability fees are still calculated based on the original `duty`, which is incorrect.

```
function setCollateralParams(address collateral, uint256 beta, uint256 rate0, bool enabled)
     external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(collateral != address(0), "AggMonetaryPolicy/invalid-address");
    require(beta > 3e5 && beta < 1e8, "AggMonetaryPolicy/invalid-beta");
    require(rate0 >= 0, "AggMonetaryPolicy/invalid-rate0");


    ilks[collateral].beta = beta;
    ilks[collateral].rate0 = rate0;
    ilks[collateral].enabled = enabled;


    emit CollateralParamsUpdated(collateral, beta, rate0, enabled);
}
```

Listing 2.13: DynamicDutyCalculator.sol

```
function drip(address token) public {
    CollateralType memory collateralType = collaterals[token];
    _checkIsLive(collateralType.live);


    bytes32 _ilk = collateralType.ilk;
    (uint256 currentDuty,) = jug.ilks(_ilk);
    uint256 duty = dutyCalculator.calculateDuty(token, currentDuty, true);
    if (duty != currentDuty) {
        _setCollateralDuty(token, duty);
    } else {
        jug.drip(_ilk);
    }
}
```

Listing 2.14: Interaction.sol

Impact The accumulated stability fees, in a period of time, is incorrect.

Suggestion Invoke the function `drip()` once updating the parameters in function `setCollateralParams()`.

Feedback from the project This is expected behaviour as well. We don't expect duty to change when we update params via `setCollateralParams()`. If needed we'll invoke `drip()` manually to update it.

2.2 Additional Recommendation

2.2.1 Redundant check in function `setCollateralParms()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In function `setCollateralParams()`, the check on line 108 is redundant. Specifically, since the type of `rate0` is `uint256`, `rate0` can never be negative.

```
105  function setCollateralParams(address collateral, uint256 beta, uint256 rate0, bool enabled)
          external onlyRole(DEFAULT_ADMIN_ROLE) {
106      require(collateral != address(0), "AggMonetaryPolicy/invalid-address");
107      require(beta > 3e5 && beta < 1e8, "AggMonetaryPolicy/invalid-beta");
108      require(rate0 >= 0, "AggMonetaryPolicy/invalid-rate0");
109
110
111      ilks[collateral].beta = beta;
112      ilks[collateral].rate0 = rate0;
113      ilks[collateral].enabled = enabled;
114
115
116      emit CollateralParamsUpdated(collateral, beta, rate0, enabled);
117  }
```

Listing 2.15: DynamicDutyCalculator.sol

Suggestion Remove this redundant check.

2.2.2 Comment mismatch with code logic

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `FixedMath0x` contract, there is an error in the comment at line 147. Specifically, within the function `_exp()`, the valid range for the parameter `x` is `EXP_MIN_VAL <= x <= 0`, but the comment incorrectly states `EXP_MIN_VAL <= x <= 1`. This inconsistency between the comment and the actual code range is misleading.

```
147  /// @dev Compute the natural exponent for a fixed-point number EXP_MIN_VAL <= ‘x‘ <= 1
148  function _exp(int256 x) internal pure returns (int256 r) {
149      if (x < EXP_MIN_VAL) {
150          // Saturate to zero below EXP_MIN_VAL.
151          return 0;
152      }
153      if (x == 0) {
154          return FIXED_1;
155      }
156      if (x > EXP_MAX_VAL) {
157          revert ExpTooLarge(x);
158      }
159  ......
160  }
```

Listing 2.16: FixedMath0x.sol

```
int256 private constant EXP_MAX_VAL = 0;
// Minimum exp argument. Notice this is related to LN_MIN_VAL (-63.875)
int256 private constant EXP_MIN_VAL = -int256(0
    x000000000000000000000000000001ff0000000000000000000000000000000);
```

Listing 2.17: FixedMath0x.sol

Suggestion Revise the comment to ensure consistency with the code implementation.

2.2.3 Potential precision loss in function `liquidationPriceForDebt()`

Status Confirmed

Introduced by `Version 1`

Description In the `Interaction` contract, the function `liquidationPriceForDebt()` is designed to calculate the price at which collateral will be liquidated. At line 478, due to Solidity's lack of floating-point support and its default behavior of rounding down in division calculations, precision loss occurs. Specifically, since all parameters involved in the calculation are of type `uint256`, there is no risk of overflow. To minimize precision loss, consider performing the division as the last operation in the calculation sequence.

```
function liquidationPriceForDebt(bytes32 ilk, uint256 ink, uint256 art) internal view returns (uint256) {
    if (ink == 0) {
        return 0; // no meaningful price if user has no debt
    }
    (, uint256 rate,,,) = vat.ilks(ilk);
    (,uint256 mat) = spotter.ilks(ilk);
    uint256 backedDebt = (art * rate / 10 ** 36) * mat;
    return backedDebt / ink;
}
```

Listing 2.18: Interaction.sol

Suggestion The division operation should be performed last in the calculation sequence.

Feedback from the project Acknowledged. This issue will be fixed in our next version.

2.2.4 Timely update role in function file()

Status Fixed in Version 2

Introduced by Version 1

Description In the DynamicDutyCalculator contract, the admin can set a new interaction address through the function file(). However, there is a failure to promptly invoke grantRole() to assign permissions to it. Specifically, only accounts with INTERACTION permissions can invoke the function calculateDuty(). Ensure that the appropriate permissions are set for the new interaction before it is used.

```
function calculateDuty(address _collateral, uint256 _currentDuty, bool _updateLastPrice) public onlyRole(INTERACTION) returns (uint256 duty) {
    Ilk storage ilk = ilks[_collateral];
    if (!ilk.enabled) {
        return _currentDuty; // if collateral not enabled for dynamic interest rate mechanism, return current duty
    }
    uint256 price = oracle.peek(lisUSD);


    // return max duty if price is too low
    if (price <= minPrice) {
        if (_updateLastPrice) {
            ilk.lastPrice = price;
        }
        return maxDuty;
    }


    // return min duty if price is too high
    if (price >= maxPrice) {
        if (_updateLastPrice) {
            ilk.lastPrice = price;
        }
```

```
            return minDuty;
        }


        // return current duty if lastPrice - 0.002 <= price <= lastPrice + 0.002
        if (price <= ilk.lastPrice + priceDeviation && price >= ilk.lastPrice - priceDeviation) {
            return _currentDuty;
        }


        if (_updateLastPrice) {
            ilk.lastPrice = price;
        }


        uint256 rate = calculateRate(price, ilk.beta, ilk.rate0);
        duty = rate + 1e27;
    }
```

Listing 2.19: DynamicDutyCalculator.sol

```
    function file(bytes32 what, address _addr) external onlyRole(DEFAULT_ADMIN_ROLE) {
        require(_addr != address(0), "AggMonetaryPolicy/zero-address-provided");


        if (what == "interaction") {
            require(interaction != _addr, "AggMonetaryPolicy/interaction-already-set");
            interaction = _addr;
        } else if (what == "lisUSD") {
            require(lisUSD != _addr, "AggMonetaryPolicy/lisUSD-already-set");
            lisUSD = _addr;
        } else if (what == "oracle") {
            require(address(oracle) != _addr, "AggMonetaryPolicy/oracle-already-set");
            oracle = IResilientOracle(_addr);
        } else revert("AggMonetaryPolicy/file-unrecognized-param");


        emit File(what, _addr);
    }
```

Listing 2.20: DynamicDutyCalculator.sol

Suggestion Revise the logic to ensure that the new interaction address is promptly granted permissions.

Feedback from the project That roles will be manually provided.

2.3 Notes

2.3.1 Potential centralization risk

Introduced by `Version 1`

Description In the DynamicDutyCalculator contract, the admin can modify key parameters (e.g., oracle) within the protocol through the function file(). If the admin account's private key is lost or maliciously exploited, it could cause significant damage to the protocol.

Feedback from the project Noted. The admin will be a multi-sig.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS