ABDK CONSULTING

SMART CONTRACT AUDIT

Risedle

Risedle

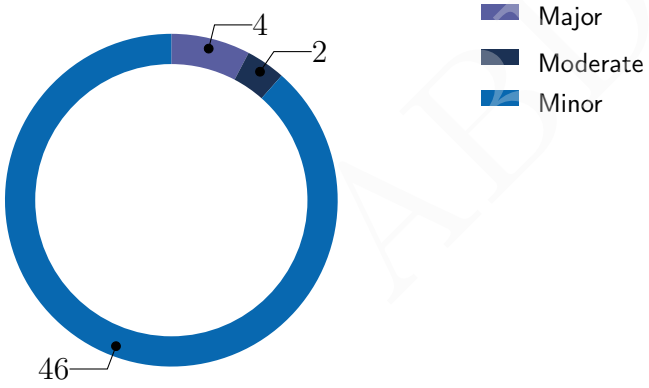
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
9th June 2022

We've been asked to review 2 files in a Github repository. We found 4 major, and a few less important issues.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Fixed
CVF-2	Minor	Procedural	Fixed
CVF-3	Minor	Procedural	Info
CVF-4	Minor	Bad datatype	Fixed
CVF-5	Minor	Bad datatype	Fixed
CVF-6	Minor	Suboptimal	Fixed
CVF-7	Minor	Bad datatype	Fixed
CVF-8	Minor	Bad datatype	Fixed
CVF-9	Minor	Bad datatype	Fixed
CVF-10	Major	Suboptimal	Fixed
CVF-11	Minor	Suboptimal	Fixed
CVF-12	Minor	Suboptimal	Fixed
CVF-13	Minor	Suboptimal	Fixed
CVF-14	Minor	Procedural	Fixed
CVF-15	Minor	Procedural	Fixed
CVF-16	Minor	Procedural	Info
CVF-17	Minor	Procedural	Fixed
CVF-18	Minor	Bad datatype	Fixed
CVF-19	Minor	Bad datatype	Fixed
CVF-20	Minor	Bad datatype	Fixed
CVF-21	Minor	Bad datatype	Fixed
CVF-22	Minor	Suboptimal	Fixed
CVF-23	Minor	Suboptimal	Fixed
CVF-24	Moderate	Suboptimal	Fixed
CVF-25	Moderate	Overflow/Underflow	Fixed
CVF-26	Minor	Suboptimal	Fixed
CVF-27	Minor	Suboptimal	Fixed

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Fixed
CVF-29	Major	Suboptimal	Info
CVF-30	Minor	Suboptimal	Fixed
CVF-31	Minor	Suboptimal	Fixed
CVF-32	Minor	Procedural	Fixed
CVF-33	Minor	Suboptimal	Fixed
CVF-34	Minor	Readability	Fixed
CVF-35	Minor	Readability	Fixed
CVF-36	Minor	Suboptimal	Fixed
CVF-37	Minor	Suboptimal	Info
CVF-38	Minor	Suboptimal	Fixed
CVF-39	Minor	Suboptimal	Fixed
CVF-40	Minor	Overflow/Underflow	Fixed
CVF-41	Minor	Suboptimal	Fixed
CVF-42	Minor	Suboptimal	Fixed
CVF-43	Minor	Suboptimal	Fixed
CVF-44	Minor	Suboptimal	Fixed
CVF-45	Minor	Suboptimal	Info
CVF-46	Minor	Bad datatype	Fixed
CVF-47	Minor	Suboptimal	Fixed
CVF-48	Minor	Bad naming	Fixed
CVF-49	Major	Suboptimal	Fixed
CVF-50	Minor	Procedural	Fixed
CVF-51	Major	Suboptimal	Fixed
CVF-52	Minor	Procedural	Fixed

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	June 8, 2022	D. Khovratovich	Initial Draft
0.2	June 8, 2022	D. Khovratovich	Minor revision
1.0	June 9, 2022	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.
We have reviewed the contracts at repository:

- RiseToken.sol
- RiseTokenFactory.sol

The fixes were provided in a new commit.

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.

- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** RiseTokenFactory.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.
Recommendation Consider specifying as "^0.8.0".
Client Comment All contracts now use ^0.8.0.

Listing 1:

```
2 pragma solidity 0.8.11;
```

3.2 CVF-2

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** RiseTokenFactory.sol

Description This pragma has no effect in Solidity 0.8.x.
Recommendation Consider removing it.
Client Comment All experimental pragma are removed on all contracts.

Listing 2:

```
3 pragma experimental ABIEncoderV2;
```

3.3 CVF-3

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** RiseTokenFactory.sol

Description We didn't review these files.
Client Comment These contracts only serve as interface and contract call forwarder to RariFuse and Uniswap.

Listing 3:

```
8 import { IfERC20 } from "./interfaces/IfERC20.sol";
  import { IRiseTokenFactory } from "./interfaces/
    ↪ IRiseTokenFactory.sol";

12 import { UniswapAdapter } from "./adapters/UniswapAdapter.sol";
  import { RariFusePriceOracleAdapter } from "./adapters/
    ↪ RariFusePriceOracleAdapter.sol";
```

3.4 CVF-4

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RiseTokenFactory.sol

Recommendation The type of this variable should be "IRiseToken[]".
Client Comment Storage data type updated.

Listing 4:

```
23 address[] public tokens;
```

3.5 CVF-5

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RiseTokenFactory.sol

Recommendation The type of both keys for this mapping should be "IfERC20". The values type for this mapping should be "IRiseToken".
Client Comment Mapping data type updated.

Listing 5:

```
24 mapping(address => mapping(address => address)) public getToken;
```

3.6 CVF-6

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseTokenFactory.sol

Description This event is emitted even if nothing actually changed.
Client Comment setFeeRecipient now revert if _newFeeRecipient is equal to existing feeRecipient. FeeRecipientUpdated only emitted when the fee recipient is updated.

Listing 6:

```
40 emit FeeRecipientUpdated(_newRecipient);
```

3.7 CVF-7

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RiseTokenFactory.sol

Recommendation The type of the "_fCollateral" and "_fDebt" arguments should be "IfERC20".
Client Comment Data type updated.

Listing 7:

```
44 function create(address _fCollateral, address _fDebt, address
   ↪ _uniswapAdapter, address _oracleAdapter) external
   ↪ onlyOwner returns (address _token) {
```

3.8 CVF-8

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RiseTokenFactory.sol

Recommendation The type of the "_uniswapAdater" argument should be "UniswapAdapter".
Client Comment Data type updated.

Listing 8:

```
44 function create(address _fCollateral, address _fDebt, address
   ↪ _uniswapAdapter, address _oracleAdapter) external
   ↪ onlyOwner returns (address _token) {
```

3.9 CVF-9

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RiseTokenFactory.sol

Recommendation The type of the "_oracleAdapter" should be "RariFusePriceOracleAdapter".
Client Comment Data type updated.

Listing 9:

```
44 function create(address _fCollateral, address _fDebt, address
   ↪ _uniswapAdapter, address _oracleAdapter) external
   ↪ onlyOwner returns (address _token) {
```

3.10 CVF-10

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseTokenFactory.sol

Description The expression "IERC20Metadata(collateral).symbol()" is calculated twice.
Recommendation Consider calculating once and reusing.
Client Comment ERC20Metadata(collateral).symbol() is cached in collateralSymbol variable.

Listing 10:

```
51 string memory tokenName = string(abi.encodePacked(IERC20Metadata
   ↪ (collateral).symbol(), " 2x Long Risedle"));
string memory tokenSymbol = string(abi.encodePacked(
   ↪ IERC20Metadata(collateral).symbol(), "RISE"));
```

3.11 CVF-11

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseTokenFactory.sol

Description Including the collateral and debt addresses into salt is redundant, as these values are already included into the creation code as a part of the contractor arguments.
Client Comment Salt is removed.

Listing 11:

```
55 bytes32 salt = keccak256(abi.encodePacked(_fCollateral, _fDebt))
   ↪ ;
```

3.12 CVF-12

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseTokenFactory.sol

Description Solidity allows creating a contract via the "CREATE2" opcode without assembly.
Recommendation Consider using plain Solidity syntax.
Client Comment Contract creation is now use 'new RiseToken' in plain solidity.

Listing 12:

```
57 _token := create2(0, add(bytecode, 32), mload(bytecode), salt)
```

3.13 CVF-13

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseTokenFactory.sol

Description This looks redundant.
Recommendation It would be cheaper to a reader to just try both variants.
Client Comment Storage write is removed and now it checks the both variant instead.

Listing 13:

```
61 getToken[_fDebt][_fCollateral] = _token; // populate mapping in
   ↪ the reverse direction
```

3.14 CVF-14

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** RiseToken.sol

Description Specifying a particular compiler version makes it harder to migrate to newer versions.
Recommendation Consider specifying as "^0.8.0".
Client Comment Solidity version updated.

Listing 14:

```
2 pragma solidity 0.8.11;
```

3.15 CVF-15

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** RiseToken.sol

Description This pragma has no effect in Solidity 0.8.x.
Recommendation Consider removing it.
Client Comment All experimental pragma are removed on all contracts.

Listing 15:

```
3 pragma experimental ABIEncoderV2;
```

3.16 CVF-16

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** RiseToken.sol

Description We didn't review these files.
Client Comment These contracts only serve as interface and contract call forwarder to RariFuse and Uniswap.

Listing 16:

```
 9 import { IRiseToken } from "./interfaces/IRiseToken.sol";
10 import { IfERC20 } from "./interfaces/IfERC20.sol";
   import { IFuseComptroller } from "./interfaces/IFuseComptroller.
     ↪ sol";
   import { IWETH9 } from "./interfaces/IWETH9.sol";

15 import { UniswapAdapter } from "./adapters/UniswapAdapter.sol";
   import { RariFusePriceOracleAdapter } from "./adapters/
     ↪ RariFusePriceOracleAdapter.sol";
```

3.17 CVF-17

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** RiseToken.sol

Description Thee variables should be declared as immutable.
Client Comment Updated to immutable storage.

Listing 17:

```
31 IWETH9                      public weth;
   RiseTokenFactory            public factory;
   UniswapAdapter              public uniswapAdapter;
   RariFusePriceOracleAdapter  public oracleAdapter;

36 ERC20   public collateral;
   ERC20   public debt;
   IfERC20 public fCollateral;
   IfERC20 public fDebt;

51 uint8 private cdecimals;
   uint8 private ddecimals;
```

3.18 CVF-18

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RiseToken.sol

Recommendation The type of this argument should be "RiseTokenFactory".
Client Comment Datatype updated.

Listing 18:

```
60 address _factory,
```

3.19 CVF-19

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RiseToken.sol

Recommendation The type of these arguments should be "IfERC20".
Client Comment Datatype updated.

Listing 19:

```
61 address _fCollateral,
   address _fDebt,
```

3.20 CVF-20

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RiseToken.sol

Recommendation The type of this argument should be "UniswapAdapter".
Client Comment Datatype updated.

Listing 20:

```
63 address _uniswapAdapter,
```

3.21 CVF-21

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RiseToken.sol

Recommendation The type of this argument should be "RariFusePriceOracleAdapter".
Client Comment Datatype updated.

Listing 21:

```
64 address _oracleAdapter
```

3.22 CVF-22

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description In ERC20 the optional property "decimals" is used by UI to render token amounts in a human-friendly way. Using this property in smart contracts is discouraged.
Recommendation Consider treating all token amounts as integers.
Client Comment Dependency on .decimals() is removed.

Listing 22:

```
75  cdecimals = collateral.decimals();
    ddecimals = debt.decimals();

332 uint256 quoteDecimals = ERC20(_quote).decimals();
```

3.23 CVF-23

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description Increasing allowance every time is suboptimal.
Recommendation Consider approving a large value once instead.
Client Comment Allowance increased at once.

Listing 23:

```
 86 collateral.safeIncreaseAllowance(address(fCollateral),
      ↪ _collateralAmount);

102 debt.safeIncreaseAllowance(address(fDebt), _repayAmount);

129 debt.safeIncreaseAllowance(address(uniswapAdapter), params.
      ↪ borrowAmount);

159 debt.safeIncreaseAllowance(address(uniswapAdapter), params.
      ↪ debtAmount);

204       collateral.safeIncreaseAllowance(address(uniswapAdapter),
            ↪ params.collateralAmount);

211       collateral.safeIncreaseAllowance(address(uniswapAdapter),
            ↪ params.collateralAmount);

217         weth.safeIncreaseAllowance(address(weth), wethLeft);

229         weth.safeIncreaseAllowance(address(uniswapAdapter),
              ↪ wethLeft);

404       weth.safeIncreaseAllowance(address(uniswapAdapter),
            ↪ wethLeftFromFlashSwap);

429 collateral.safeTransferFrom(msg.sender, address(this), _amountIn
      ↪ );

436 debt.safeIncreaseAllowance(address(uniswapAdapter), borrowAmount
      ↪ );

442       debt.safeIncreaseAllowance(address(fDebt), repayAmount);

449 weth.safeIncreaseAllowance(address(weth), _amountOut);

474 weth.safeIncreaseAllowance(address(uniswapAdapter), msg.value);
```

3.24 CVF-24

- **Severity** Moderate
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Recommendation Should be "||" instead of "&&". Currently, an error is reported only when both market statuses are not zero.

Listing 24:

```
124 if (marketStatus[0] != 0 && marketStatus[1] != 0) revert
    ↪ FuseError(marketStatus[0]);
```

3.25 CVF-25

- **Severity** Moderate
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** RiseToken.sol

Description Underflow could happen here causing the transaction to revert.
Recommendation Consider properly handling the situation when _wethAmount < wethAmountFromBorrow. For example, transfer the excess WETH amount back to the initializer, either as WETH or as plain ether.
Client Comment The _wethAmount < wethAmountFromBorrow is now handled properly. Excess WETH is sent to initializer .

Listing 25:

```
133 uint256 owedWETH = _wethAmount - wethAmountFromBorrow;

163 uint256 owedWETH = _wethAmount - wethAmountFromBorrow;
```

3.26 CVF-26

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description This should be executed only when excessETH > 0.
Client Comment Now ETH is sent only if excessETH > 0.

Listing 26:

```
138 (bool sent, ) = params.initializer.call{value: excessETH}("");
    if (!sent) revert FailedToSendETH(params.initializer, excessETH)
    ↪ ;

169     (bool sent, ) = params.buyer.call{value: excessETH}("");
170     if (!sent) revert FailedToSendETH(params.buyer, excessETH);
```

3.27 CVF-27

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description This should be executed only when owedWETH > 0.
Client Comment Now ETH is converted to WETH only if owedWETH > 0.

Listing 27:

```
142 weth.deposit{ value: owedWETH }(); // Wrap the ETH to WETH

171     weth.deposit{ value: owedWETH }();
```

3.28 CVF-28

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description This should be executed only when _wethAmount > 0.
Client Comment Now the flashloan is only repay when _wethAmount > 0.

Listing 28:

```
143 weth.safeTransfer(address(uniswapAdapter), _wethAmount);

182 weth.safeTransfer(address(uniswapAdapter), _wethAmount);
```

3.29 CVF-29

- **Severity** Major
- **Status** Info
- **Category** Suboptimal
- **Source** RiseToken.sol

Description For a single swap, the "in" tokens are transferred 5 times: two times here, two times inside the Uniswap adapter, and one more time inside the Uniswap router. This consumes lots of gas.
Recommendation Consider calculating the exact input amount to be swapped and transferring this amount once directly from the buyer to the Uniswap pair.
Client Comment This is no issue because RiseToken only runs on L2.

Listing 29:

```
173 params.tokenIn.safeTransferFrom(params.buyer, address(this),
    ↪ params.amountInMax);

177     params.tokenIn.safeTransfer(params.buyer, params.amountInMax
        ↪ - amountIn);
```

3.30 CVF-30

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description This should only be executed when collateralLeft > 0.
Client Comment If collateralLeft = 0 now reverted.

Listing 30:

```
208 collateral.safeTransfer(params.recipient, collateralLeft);
```

3.31 CVF-31

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description This should be executed only when wethLeft > 0.
Client Comment wethOut and amountOut is now checked.

Listing 31:

```
217     weth.safeIncreaseAllowance(address(weth), wethLeft);
        weth.withdraw(wethLeft);
        (bool sent, ) = params.recipient.call{value: wethLeft}("");
220     if (!sent) revert FailedToSendETH(params.recipient, wethLeft
          ↪ );

228 if (address(params.tokenOut) != address(0) && (address(params.
      ↪ tokenOut) != address(debt))) {
        weth.safeIncreaseAllowance(address(uniswapAdapter), wethLeft
          ↪ );
230     uint256 amountOut = uniswapAdapter.swapExactWETHForTokens(
          ↪ address(params.tokenOut), wethLeft, params.
          ↪ amountOutMin);
        params.tokenOut.safeTransfer(params.recipient, amountOut);
    }
```

3.32 CVF-32

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** RiseToken.sol

Description There are no range checks for the arguments.
Recommendation Consider adding appropriate checks.
Client Comment Input is now checked.

Listing 32:

```
248 function setParams(uint256 _minLeverageRatio, uint256
    ↪ _maxLeverageRatio, uint256 _step, uint256 _discount,
    ↪ uint256 _newMaxBuy) external onlyOwner {
```

3.33 CVF-33

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Recommendation This could be simplified as: if (isInitialized) revert (...);

Listing 33:

```
259 if (isInitialized == true) revert AlreadyInitialized();
```

3.34 CVF-34

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** RiseToken.sol

Recommendation Should be "} else if (...) {" for readability.

Listing 34:

```
276     return;
}

if (flashSwapType == FlashSwapType.Buy) {
```

3.35 CVF-35

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** RiseToken.sol

Recommendation Should be "} else if (...) {" for readability.

Listing 35:

```
282 }

    if (flashSwapType == FlashSwapType.Sell) {
```

3.36 CVF-36

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Recommendation Should be "} else revert (...);" for completeness.

Listing 36:

```
287 }
```

3.37 CVF-37

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** RiseToken.sol

Description The precision of the returned values for these functions depends of the numbers of decimals of underlying tokens. This could cause problems with tokens that have little decimals.
Recommendation Consider using the same precision for all tokens, say 18 decimals.
Client Comment The _dps and _cps base units is designed to follow the base units of the collateral and debt tokens.

Listing 37:

```
297 function collateralPerShare() public view returns (uint256 _cps)
        ↪ {

303 function debtPerShare() public view returns (uint256 _dps) {
```

3.38 CVF-38

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Recommendation Consider extracting these checks to a modifier.
Client Comment Modifier whenInitialized is added.

Listing 38:

```
298 if (!isInitialized) return 0;

304 if (!isInitialized) return 0;

310 if (!isInitialized) return 0;

342 if (!isInitialized) return 0;

348 if (!isInitialized) return 0;

359 if (!isInitialized) revert NotInitialized();

383 if (!isInitialized) revert NotInitialized();
```

3.39 CVF-39

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description The value "10**cdecimals" is calculated every time.
Recommendation Consider calculating once in constructor and storing in an immutable variable.
Client Comment The base units is now abstracted.

Listing 39:

```
299 _cps = (totalCollateral * (10**cdecimals)) / totalSupply();

305 _dps = (totalDebt * (10**cdecimals)) / totalSupply();

313 uint256 collateralAmount = (_shares * collateralPerShare()) /
      ↪ (10**cdecimals);
    uint256 debtAmount = (_shares * debtPerShare()) / (10**cdecimals
      ↪ );

321 uint256 collateralValue = (collateralAmount * cPrice) / (10**
      ↪ cdecimals);

343 _nav = value(10**cdecimals);

350 uint256 collateralValue = (collateralPerShare() *
      ↪ collateralPrice) / (10**cdecimals);

370     collateralAmount: (newShares * collateralPerShare()) / (10**
          ↪ cdecimals),
        debtAmount: (newShares * debtPerShare()) / (10**cdecimals),

393     collateralAmount: (newShares * collateralPerShare()) / (10**
          ↪ cdecimals),
        debtAmount: (newShares * debtPerShare()) / (10**cdecimals),

432 uint256 borrowAmount = ((step * value((10**cdecimals), address(
      ↪ debt)) / 1e18) * totalSupply()) / (10**cdecimals);

466 _amountOut = (msg.value * (10**cdecimals)) / price;

473 uint256 repayAmount = ((step * value((10**cdecimals), address(
      ↪ debt)) / 1e18) * totalSupply()) / (10**cdecimals);
```

3.40 CVF-40

- **Severity** Minor
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** RiseToken.sol

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculation overflows.
Recommendation Consider using the muldiv function as described here: https://2.com/21/muldiv/index.html or some other approach that prevents phantom overflow.
Client Comment All computation now use FixedPointMathLib from Rari Capital Solmate library.

Listing 40:

```
299 _cps = (totalCollateral * (10**cdecimals)) / totalSupply();

305 _dps = (totalDebt * (10**cdecimals)) / totalSupply();

313 uint256 collateralAmount = (_shares * collateralPerShare()) /
      ↪ (10**cdecimals);
    uint256 debtAmount = (_shares * debtPerShare()) / (10**cdecimals
      ↪ );

321 uint256 collateralValue = (collateralAmount * cPrice) / (10**
      ↪ cdecimals);
    uint256 debtValue = (debtAmount * dPrice) / (10**ddecimals);

334 uint256 amountInETH = (valueInETH * 1e18) / quotePrice;

337 _value = (amountInETH * (10**quoteDecimals)) / 1e18;

350 uint256 collateralValue = (collateralPerShare() *
      ↪ collateralPrice) / (10**cdecimals);
    _lr = (collateralValue * 1e18) / nav();

362 uint256 fee = ((fees * _shares) / 1e18);

370     collateralAmount: (newShares * collateralPerShare()) / (10**
          ↪ cdecimals),
        debtAmount: (newShares * debtPerShare()) / (10**cdecimals),

385 uint256 fee = ((fees * _shares) / 1e18);

393     collateralAmount: (newShares * collateralPerShare()) / (10**
          ↪ cdecimals),
        debtAmount: (newShares * debtPerShare()) / (10**cdecimals),
```

(... 422, 432, 465, 473)

3.41 CVF-41

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description Silently returning zero when a functions is called on an uninitialized smart contract is error-prone, as the calling contract may forget to perform a check.
Recommendation Consider reverting instead.
Client Comment Now reverted in the modifier whenInitialized.

Listing 41:

```
298 if (!isInitialized) return 0;

304 if (!isInitialized) return 0;

310 if (!isInitialized) return 0;

342 if (!isInitialized) return 0;

348 if (!isInitialized) return 0;
```

3.42 CVF-42

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Recommendation This formula is equivalent to: _shares * totalCollateral * 10**cdecimals / totalSupply / 10**cdecimals and could be reduced to: _shares * totalCollateral / totalSupply.
Client Comment The collateral amount calculation is now simplified.

Listing 42:

```
313 uint256 collateralAmount = (_shares * collateralPerShare()) /
    ↪ (10**cdecimals);
```

3.43 CVF-43

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Recommendation This formula is equivalent to: _shares * totalDebt * 10**cdecimals / totalSupply / 10**cdecimals and could be reduced to: _shares * totalDebt / totalSupply.
Client Comment The debt amount calculation is now simplified.

Listing 43:

```
314 uint256 debtAmount = (_shares * debtPerShare()) / (10**cdecimals
    ↪ );
```

3.44 CVF-44

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description The value "10**ddecimals" is calculated every time.
Recommendation Consider calculating once in the constructor and storing in an immutable variable.
Client Comment (10**ddecimals) is no longer used, simplified as base units.

Listing 44:

```
322 uint256 debtValue = (debtAmount * dPrice) / (10**ddecimals);
```

3.45 CVF-45

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** RiseToken.sol

Description This cannot return a negative value and will just revert in case debt exceeds collateral..
Recommendation Consider supporting negative values.
Client Comment The value can't be negative because the collateral value will always > debt value due too rebalancing mechanism.

Listing 45:

```
325 _value = collateralValue - debtValue;
```

3.46 CVF-46

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** RiseToken.sol

Recommendation The value "1e18" should be a named constant.
Client Comment 1e18 replaced with 'ether'.

Listing 46:

```
334 uint256 amountInETH = (valueInETH * 1e18) / quotePrice;

337 _value = (amountInETH * (10**quoteDecimals)) / 1e18;

351 _lr = (collateralValue * 1e18) / nav();

362 uint256 fee = ((fees * _shares) / 1e18);

385 uint256 fee = ((fees * _shares) / 1e18);

422 price += (discount * price) / 1e18;
    _amountOut = (_amountIn * price) / (1e18);

432 uint256 borrowAmount = ((step * value((10**cdecimals), address(
      ↪ debt)) / 1e18) * totalSupply()) / (10**cdecimals);

465 price -= (discount * price) / 1e18;

473 uint256 repayAmount = ((step * value((10**cdecimals), address(
      ↪ debt)) / 1e18) * totalSupply()) / (10**cdecimals);
```

3.47 CVF-47

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description The value "10**quoteDecimals" is calculated every time.
Recommendation Consider calculating once in the constructor and storing in an immutable variable.
Client Comment (10**ddecimals) is no longer used, simplified as base units.

Listing 47:

```
337 _value = (amountInETH * (10**quoteDecimals)) / 1e18;
```

3.48 CVF-48

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** RiseToken.sol

Description The name is confusing, as this function actually calculates the NAV per share value, rather than just NAV value.
Client Comment nav() is renamed to price().

Listing 48:

```
341 function nav() public view returns (uint256 _nav) {
```

3.49 CVF-49

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description This function calculates the leverage ratio using the per-share collateral value and per-share NAV.
Recommendation It would be more efficient to use total collateral value and total NAV instead. Also, this would make the result more precise.
Client Comment Leverage ratio calculation is simplified.

Listing 49:

```
347 function leverageRatio() public view returns (uint256 _lr) {
```

3.50 CVF-50

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** RiseToken.sol

Recommendation Brackets are redundant here.
Client Comment Fee calculation now use mulWadDown.

Listing 50:

```
362 uint256 fee = ((fees * _shares) / 1e18);

385 uint256 fee = ((fees * _shares) / 1e18);
```

3.51 CVF-51

- **Severity** Major
- **Status** Fixed
- **Category** Suboptimal
- **Source** RiseToken.sol

Description This formula calculates the share price in debt token terms, dividing the total collateral and debt amount by the total supply, and then multiplies the result by total supply. This is suboptimal and could be optimized.
Recommendation Consider refactoring.
Client Comment Calculation is simplified.

Listing 51:

```
432 uint256 borrowAmount = ((step * value((10**cdecimals), address(
      ↪ debt)) / 1e18) * totalSupply()) / (10**cdecimals);

473 uint256 repayAmount = ((step * value((10**cdecimals), address(
      ↪ debt)) / 1e18) * totalSupply()) / (10**cdecimals);
```

3.52 CVF-52

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** RiseToken.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.
Client Comment Comment added.

Listing 52:

```
485 receive() external payable {}
```