# ABDK CONSULTING

SMART CONTRACT AUDIT

## MORET

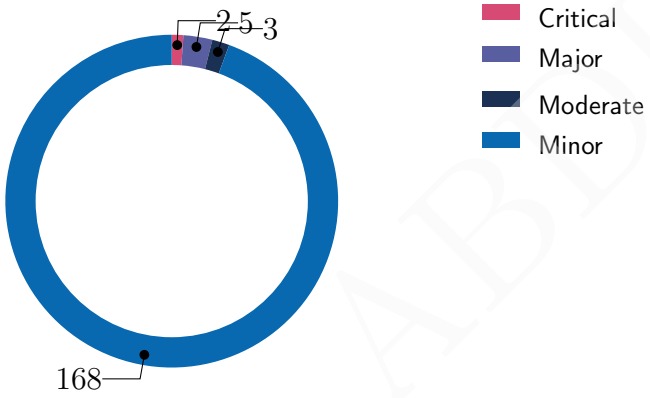
**Solidity**

abdk.consulting

# SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
12th January 2022

We've been asked to review certain files in a Github repo. We found 2 critical, 5 major, and a few less important issues. All critical and major issues were fixed.



# Findings

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

| ID | Severity | Category | Status |
|---|---|---|---|
| CVF-28 | Minor | Bad datatype | Fixed |
| CVF-29 | Minor | Overflow/Underflow | Fixed |
| CVF-30 | Minor | Procedural | Info |
| CVF-31 | Minor | Readability | Fixed |
| CVF-32 | Minor | Procedural | Fixed |
| CVF-33 | Minor | Procedural | Fixed |
| CVF-34 | Minor | Bad naming | Fixed |
| CVF-35 | Minor | Procedural | Fixed |
| CVF-36 | Minor | Procedural | Fixed |
| CVF-37 | Minor | Documentation | Fixed |
| CVF-38 | Minor | Procedural | Fixed |
| CVF-39 | Minor | Procedural | Fixed |
| CVF-40 | Minor | Readability | Fixed |
| CVF-41 | Minor | Bad naming | Fixed |
| CVF-42 | Minor | Procedural | Fixed |
| CVF-43 | Minor | Procedural | Fixed |
| CVF-44 | Minor | Procedural | Fixed |
| CVF-45 | Minor | Procedural | Fixed |
| CVF-46 | Minor | Bad datatype | Fixed |
| CVF-47 | Minor | Readability | Fixed |
| CVF-48 | Minor | Bad datatype | Fixed |
| CVF-49 | Minor | Bad datatype | Info |
| CVF-50 | Minor | Bad datatype | Fixed |
| CVF-51 | Critical | Flaw | Fixed |
| CVF-52 | Minor | Suboptimal | Fixed |
| CVF-53 | Minor | Procedural | Fixed |
| CVF-54 | Major | Unclear behavior | Fixed |
| CVF-55 | Minor | Documentation | Info |
| CVF-56 | Moderate | Unclear behavior | Info |
| CVF-57 | Minor | Suboptimal | Info |

| ID | Severity | Category | Status |
|---|---|---|---|
| CVF-58 | Minor | Suboptimal | Fixed |
| CVF-59 | Minor | Suboptimal | Fixed |
| CVF-60 | Minor | Suboptimal | Fixed |
| CVF-61 | Major | Flaw | Fixed |
| CVF-62 | Minor | Readability | Fixed |
| CVF-63 | Minor | Bad datatype | Fixed |
| CVF-64 | Minor | Bad datatype | Fixed |
| CVF-65 | Minor | Bad datatype | Fixed |
| CVF-66 | Major | Procedural | Fixed |
| CVF-67 | Minor | Suboptimal | Fixed |
| CVF-68 | Minor | Procedural | Fixed |
| CVF-69 | Minor | Procedural | Fixed |
| CVF-70 | Minor | Procedural | Info |
| CVF-71 | Minor | Readability | Fixed |
| CVF-72 | Minor | Bad naming | Fixed |
| CVF-73 | Minor | Bad naming | Fixed |
| CVF-74 | Minor | Readability | Fixed |
| CVF-75 | Minor | Unclear behavior | Info |
| CVF-76 | Minor | Suboptimal | Fixed |
| CVF-77 | Minor | Suboptimal | Fixed |
| CVF-78 | Major | Flaw | Fixed |
| CVF-79 | Minor | Suboptimal | Fixed |
| CVF-80 | Minor | Bad naming | Fixed |
| CVF-81 | Minor | Suboptimal | Info |
| CVF-82 | Minor | Suboptimal | Fixed |
| CVF-83 | Minor | Readability | Fixed |
| CVF-84 | Minor | Overflow/Underflow | Fixed |
| CVF-85 | Minor | Suboptimal | Fixed |
| CVF-86 | Minor | Readability | Fixed |
| CVF-87 | Minor | Unclear behavior | Info |

| ID | Severity | Category | Status |
|---|---|---|---|
| CVF-88 | Minor | Procedural | Fixed |
| CVF-89 | Minor | Suboptimal | Fixed |
| CVF-90 | Moderate | Overflow/Underflow | Fixed |
| CVF-91 | Minor | Suboptimal | Fixed |
| CVF-92 | Minor | Suboptimal | Info |
| CVF-93 | Major | Readability | Fixed |
| CVF-94 | Minor | Suboptimal | Info |
| CVF-95 | Minor | Readability | Fixed |
| CVF-96 | Minor | Overflow/Underflow | Fixed |
| CVF-97 | Minor | Procedural | Fixed |
| CVF-98 | Minor | Suboptimal | Fixed |
| CVF-99 | Minor | Procedural | Fixed |
| CVF-100 | Minor | Bad datatype | Fixed |
| CVF-101 | Minor | Bad datatype | Fixed |
| CVF-102 | Minor | Bad datatype | Fixed |
| CVF-103 | Minor | Bad datatype | Fixed |
| CVF-104 | Minor | Suboptimal | Fixed |
| CVF-105 | Minor | Flaw | Fixed |
| CVF-106 | Minor | Documentation | Fixed |
| CVF-107 | Minor | Unclear behavior | Fixed |
| CVF-108 | Minor | Unclear behavior | Fixed |
| CVF-109 | Minor | Suboptimal | Fixed |
| CVF-110 | Minor | Suboptimal | Fixed |
| CVF-111 | Minor | Unclear behavior | Info |
| CVF-112 | Minor | Suboptimal | Fixed |
| CVF-113 | Minor | Suboptimal | Fixed |
| CVF-114 | Minor | Suboptimal | Fixed |
| CVF-115 | Minor | Bad datatype | Fixed |
| CVF-116 | Minor | Readability | Fixed |
| CVF-117 | Minor | Suboptimal | Fixed |

| ID | Severity | Category | Status |
|---|---|---|---|
| CVF-118 | Minor | Suboptimal | Fixed |
| CVF-119 | Minor | Suboptimal | Fixed |
| CVF-120 | Minor | Suboptimal | Fixed |
| CVF-121 | Minor | Suboptimal | Fixed |
| CVF-122 | Minor | Suboptimal | Fixed |
| CVF-123 | Minor | Readability | Fixed |
| CVF-124 | Minor | Readability | Fixed |
| CVF-125 | Minor | Suboptimal | Fixed |
| CVF-126 | Minor | Overflow/Underflow | Fixed |
| CVF-127 | Minor | Suboptimal | Fixed |
| CVF-128 | Minor | Readability | Fixed |
| CVF-129 | Minor | Suboptimal | Info |
| CVF-130 | Minor | Readability | Fixed |
| CVF-131 | Minor | Suboptimal | Info |
| CVF-132 | Minor | Suboptimal | Info |
| CVF-133 | Minor | Suboptimal | Info |
| CVF-134 | Minor | Suboptimal | Fixed |
| CVF-135 | Minor | Procedural | Fixed |
| CVF-136 | Minor | Suboptimal | Fixed |
| CVF-137 | Minor | Suboptimal | Fixed |
| CVF-138 | Minor | Suboptimal | Fixed |
| CVF-139 | Minor | Suboptimal | Fixed |
| CVF-140 | Minor | Suboptimal | Fixed |
| CVF-141 | Minor | Bad naming | Info |
| CVF-142 | Minor | Suboptimal | Fixed |
| CVF-143 | Minor | Procedural | Fixed |
| CVF-144 | Minor | Procedural | Fixed |
| CVF-145 | Minor | Suboptimal | Fixed |
| CVF-146 | Minor | Bad datatype | Fixed |
| CVF-147 | Minor | Suboptimal | Fixed |

| ID | Severity | Category | Status |
|---|---|---|---|
| CVF-148 | Minor | Suboptimal | Fixed |
| CVF-149 | Minor | Unclear behavior | Fixed |
| CVF-150 | Minor | Suboptimal | Fixed |
| CVF-151 | Minor | Unclear behavior | Info |
| CVF-152 | Minor | Procedural | Fixed |
| CVF-153 | Minor | Suboptimal | Fixed |
| CVF-154 | Minor | Readability | Fixed |
| CVF-155 | Minor | Readability | Fixed |
| CVF-156 | Minor | Suboptimal | Fixed |
| CVF-157 | Minor | Readability | Fixed |
| CVF-158 | Minor | Overflow/Underflow | Fixed |
| CVF-159 | Minor | Suboptimal | Fixed |
| CVF-160 | Minor | Overflow/Underflow | Fixed |
| CVF-161 | Minor | Suboptimal | Fixed |
| CVF-162 | Moderate | Suboptimal | Info |
| CVF-163 | Minor | Suboptimal | Fixed |
| CVF-164 | Minor | Suboptimal | Fixed |
| CVF-165 | Minor | Suboptimal | Info |
| CVF-166 | Minor | Suboptimal | Info |
| CVF-167 | Minor | Suboptimal | Fixed |
| CVF-168 | Minor | Suboptimal | Fixed |
| CVF-169 | Minor | Suboptimal | Info |
| CVF-170 | Minor | Suboptimal | Fixed |
| CVF-171 | Minor | Procedural | Fixed |
| CVF-172 | Minor | Unclear behavior | Fixed |
| CVF-173 | Minor | Suboptimal | Fixed |
| CVF-174 | Minor | Procedural | Fixed |
| CVF-175 | Minor | Suboptimal | Fixed |
| CVF-176 | Minor | Suboptimal | Fixed |
| CVF-177 | Minor | Suboptimal | Fixed |

| ID | Severity | Category | Status |
| --- | --- | --- | --- |
| CVF-178 | Minor | Unclear behavior | Fixed |

# Contents

# 1 Document properties

## Version

| Version | Date | Author | Description |
|---|---|---|---|
| 0.1 | January 11, 2022 | D. Khovratovich | Initial Draft |
| 0.2 | January 12, 2022 | D. Khovratovich | Minor revision |
| 1.0 | January 12, 2022 | D. Khovratovich | Release |
| 1.1 | January 12, 2022 | D. Khovratovich | Date fix |
| 2.0 | January 12, 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 repository with the following files:

- Exchange.sol
- MarketLibrary.sol
- MoretInterfaces.sol
- MoretMarketMaker.sol
- OptionLibrary.sol
- OptionVault.sol
- VolatilityChain.sol
- VolatilityToken.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** Exchange.sol

**Description** Should be "^0.8.0".

Listing 1:

```
2 pragma solidity 0.8.10;
```

## 3.2 CVF-2

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

**Description** The type of this variable should be "OptionVault".
**Client Comment** This variable is replaced with external functions.

Listing 2:

```
13 address public vaultAddress;
```

## 3.3 CVF-3

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

**Description** The type of this variable should be "MoretMarketMaker".
**Client Comment** This variable is replaced with external functions.

Listing 3:

```
14 address public marketMakerAddress;
```

## 3.4 CVF-4

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

**Description** The type of this mapping should be: mapping (uint256 => VolatilityToken)

Listing 4:

```
15 mapping(uint256=>address) public volTokenAddressList;
```

## 3.5 CVF-5

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

**Description** These variables should be declared as immutable.
**Client Comment** Added constant modifier for the variables.

Listing 5:

```
13 address public vaultAddress;
   address public marketMakerAddress;

17 MoretMarketMaker internal marketMaker;
   OptionVault internal optionVault;
```

## 3.6 CVF-6

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

**Description** These variables duplicate the public variables "vaultAddress" and "marketMakerAddress".
**Recommendation** Consider removing.
**Client Comment** Removed the address variables.

Listing 6:

```
17 MoretMarketMaker internal marketMaker;
   OptionVault internal optionVault;
```

## 3.7 CVF-7

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

**Description** This value could be rendered as "1.8e18".
**Client Comment** Fixed and renamed 'volCapacityFactor'.

Listing 7:

```
20 uint256 public volRiskPremiumMaxRatio= 18 * (10 ** 17);
```

## 3.8 CVF-8

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

**Description** The types of the arguments should be "MoretMarketMaker" and "OptionVault" respectively.

Listing 8:

```
24 constructor( address _marketMakerAddress,address _vaultAddress){
```

## 3.9 CVF-9

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

**Description** The semantics of the returned values is unclear.
**Recommendation** Consider giving descriptive names to the returned values and/or adding a documentation comment.
**Client Comment** Created extra function 'calcOptionCost' with clearer semantics.

Listing 9:

```
32 function calcCost(uint256 _tenor, uint256 _strike, uint256
   ↪ _amount, OptionLibrary.PayoffType _poType, OptionLibrary.
   ↪ OptionSide _side) public view returns(uint256 , uint256 ,
   ↪ uint256 ){
```

## 3.10 CVF-10

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

**Description** This commented line should be removed.

Listing 10:

```
38 // uint256 _adjustedStrike = OptionLibrary.adjustStrike(_strike,
   ↪  _poType, _side, marketMaker.swapSlippage(), loanInterest)
   ↪ ;
```

## 3.11 CVF-11

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

**Description** Inside these calls the number of funding token decimals is obtained.
**Recommendation** Consider obtaining it once in the constructor and storing in an internal variable for future use.
**Client Comment** Added internal variables for storing decimals in Exchange, OptionVault and MoretMarketMaker.

Listing 11:

```
40 _premium = MarketLibrary.cvtDecimals(_premium, optionVault.
   ↪ funding());
_cost = MarketLibrary.cvtDecimals(_cost, optionVault.funding())
   ↪ ;}
```

## 3.12 CVF-12

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

**Description** Querying the option vault for the funding token every time is suboptimal.
**Recommendation** Consider querying one in the constructor and storing in an internal variable.
**Client Comment** Stored in internal variable 'fundingToken'.

Listing 12:

```
40  _premium = MarketLibrary.cvtDecimals(_premium, optionVault.
        ↪ funding());
    _cost = MarketLibrary.cvtDecimals(_cost, optionVault.funding())
        ↪ ;}

71  require(ERC20(optionVault.funding()).transferFrom(msg.sender,
        ↪ marketMakerAddress, _payInCost), 'Failed payment.');

88  require(ERC20(optionVault.funding()).transferFrom(msg.sender,
        ↪ address(this), _premium), "vol payment error");

96  require(ERC20(optionVault.funding()).balanceOf(address(this))>=
        ↪ _premium, "insufficient usdc in exchange.");

99  require(ERC20(optionVault.funding()).transfer(msg.sender,
        ↪ _premium), "payment error");

106 require(ERC20(optionVault.funding()).balanceOf(address(this))>=
        ↪ _premium, "insufficient usdc in exchange.");

110 require(ERC20(optionVault.funding()).transfer(marketMakerAddress
        ↪ , _premium), 'payment error');

120 require(ERC20(optionVault.funding()).transferFrom(msg.sender,
        ↪ marketMakerAddress, _cost), 'payment error');
    require(ERC20(optionVault.funding()).transferFrom(msg.sender,
        ↪ address(this), _premium), 'payment error');
```

## 3.13 CVF-13

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

**Description** Fixed point multiplication and division is implemented multiple times.
**Recommendation** Consider extracting into utility functions.
**Client Comment** Moved to FullMath and use it for uint256.

Listing 13:

```
46 require(MulDiv(_amount, _price, OptionLibrary.Multiplier())<=
   ↪ marketMaker.calcCapital(true,false),'insufficient capital
   ↪ ');

54 int256 _newGamma = _currentGamma + int256(MulDiv(OptionLibrary.
   ↪ calcGamma(_price, _strike, _vol), _amount, OptionLibrary.
   ↪ Multiplier() )) * (_side==OptionLibrary.OptionSide.Sell?
   ↪ -1: int(1));
   uint256 _K = MulDiv(_vol, volRiskPremiumMaxRatio, OptionLibrary.
   ↪ Multiplier());

60 if(_input < 0){_capacity += int256(MulDiv(uint256(-_input),
   ↪ OptionLibrary.Multiplier(), _max));}
   if(_input > 0){ _capacity -= int256(MulDiv(uint256(_input) ,
   ↪ OptionLibrary.Multiplier(), _max));}

63 return int256(MulDiv(_constant, OptionLibrary.Multiplier(),
   ↪ uint256(_capacity))) - int256(_constant);}

82 _volAmount = MarketLibrary.cvtDecimals(MulDiv(_premium,
   ↪ OptionLibrary.Multiplier(), _vol), volTokenAddressList[
   ↪ _tenor]);}
```

## 3.14 CVF-14

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

**Description** Querying the multiplier from the option library each time is suboptimal.
**Recommendation** Consider querying once in the constructor and storing in an internal variable.
**Client Comment** Saved in internal constant variable.

Listing 14:

```
46 require(MulDiv(_amount, _price, OptionLibrary.Multiplier())<=
   ↪ marketMaker.calcCapital(true,false),'insufficient capital
   ↪ ');

54 int256 _newGamma = _currentGamma + int256(MulDiv(OptionLibrary.
   ↪ calcGamma(_price, _strike, _vol), _amount, OptionLibrary.
   ↪ Multiplier() )) * (_side==OptionLibrary.OptionSide.Sell?
   ↪ -1: int(1));
   uint256 _K = MulDiv(_vol, volRiskPremiumMaxRatio, OptionLibrary.
   ↪ Multiplier());

59 int256 _capacity = int256(OptionLibrary.Multiplier()); //
   ↪ capacity should be in (0,2)
60 if(_input < 0){_capacity += int256(MulDiv(uint256(-_input),
   ↪ OptionLibrary.Multiplier(), _max));}
   if(_input > 0){ _capacity -= int256(MulDiv(uint256(_input) ,
   ↪ OptionLibrary.Multiplier(), _max));}
   require((_capacity>=0) && (_capacity <= int256(2 * OptionLibrary
   ↪ .Multiplier())),"Capacity breached.");
   return int256(MulDiv(_constant, OptionLibrary.Multiplier(),
   ↪ uint256(_capacity))) - int256(_constant);}

82 _volAmount = MarketLibrary.cvtDecimals(MulDiv(_premium,
   ↪ OptionLibrary.Multiplier(), _vol), volTokenAddressList[
   ↪ _tenor]);}
```

## 3.15 CVF-15

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

**Description** Overflow possible when converting to "int256".
**Recommendation** Consider using safe conversion.
**Client Comment** All replaced with 'SafeCast.toInt256'.

Listing 15:

```
48 require((int256(_vol)+_riskPremium) > 0,"Incorrect vol premium")
     ↪ ;
   _vol = uint256(int256(_vol)+_riskPremium);}

54 int256 _newGamma = _currentGamma + int256(MulDiv(OptionLibrary.
     ↪ calcGamma(_price, _strike, _vol), _amount, OptionLibrary.
     ↪ Multiplier() )) * (_side==OptionLibrary.OptionSide.Sell?
     ↪ -1: int(1));

60 if(_input < 0){_capacity += int256(MulDiv(uint256(-_input),
     ↪ OptionLibrary.Multiplier(), _max));}
   if(_input > 0){ _capacity -= int256(MulDiv(uint256(_input) ,
     ↪ OptionLibrary.Multiplier(), _max));}
   require((_capacity>=0) && (_capacity <= int256(2 * OptionLibrary
     ↪ .Multiplier())),"Capacity breached.");
   return int256(MulDiv(_constant, OptionLibrary.Multiplier(),
     ↪ uint256(_capacity))) - int256(_constant);}
```

## 3.16 CVF-16

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

**Description** This event is redundant, as the "transferFrom" and "mint" calls emit events containing enough information.
**Client Comment** Removed.

Listing 16:

```
90 emit volatilityTokenBought(msg.sender, block.timestamp, _tenor,
     ↪ _volAmount, _premium);}
```

## 3.17 CVF-17

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

**Description** These checks are redundant as the consequent "transfer" calls will fail anyway in case of insufficient balance.
**Client Comment** Removed.

Listing 17:

```
96  require(ERC20(optionVault.funding()).balanceOf(address(this))>=
    ↪ _premium, "insufficient usdc in exchange.");

106 require(ERC20(optionVault.funding()).balanceOf(address(this))>=
    ↪ _premium, "insufficient usdc in exchange.");
```

## 3.18 CVF-18

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

**Description** This event is redundant, as the "burn" and "transfer" calls emit events containing enough information.
**Client Comment** Removed.

Listing 18:

```
100 emit volatilityTokenSold(msg.sender, block.timestamp, _tenor,
    ↪ _volAmount, _premium);}
```

## 3.19 CVF-19

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

**Description** These functions should emit some events.
**Client Comment** Added two new events 'volTokenAdded' and 'volTokenRemoved'.

Listing 19:

```
127 function addVolToken(uint256 _tenor, address _tokenAddress)
    ↪ external onlyRole(ADMIN_ROLE){

130 function removeVolToken(uint256 _tenor) external onlyRole(
    ↪ ADMIN_ROLE){ volTokenAddressList[_tenor] = address(0);}
```

## 3.20 CVF-20

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

**Description** The expression "volTokenAddressList[_tenor]" is calculated several times.
**Recommendation** Consider calculating once and reusing.
**Client Comment** This line has been modified so not relevant now.

Listing 20:

```
128 require(VolatilityToken(volTokenAddressList[_tenor]).tenor()==
    ↪ _tenor && VolatilityToken(volTokenAddressList[_tenor]).
    ↪ tokenHash()==optionVault.tokenHash(), 'mismatched token
    ↪ address');
volTokenAddressList[_tenor] = _tokenAddress; }
```

## 3.21 CVF-21

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** Exchange.sol

**Description** The "volTokenAddressList" mapping is empty initially, and this function is the only way to populate it, however, this check calls the "tenor" and "tokenHash" functions on the current volatility token, which is initially zero, thus will not get meaningful result and most probably will fail.
**Recommendation** It should be "_tokenAddress" here instead of "volTokenAddressList[_tender]".
**Client Comment** I've replaced 'volTokenAddressList[_tenor]' with '_tokenAddress', which should be the correct way of checking the tenor and token hash properties.

Listing 21:

```
128   require(VolatilityToken(volTokenAddressList[_tenor]).tenor()==
      ↪ _tenor && VolatilityToken(volTokenAddressList[_tenor]).
      ↪ tokenHash()==optionVault.tokenHash(), 'mismatched token
      ↪ address');

132 function resetLoanRate(uint256 _loanInterest) external onlyRole(
    ↪ ADMIN_ROLE){ loanInterest = _loanInterest;}
function resetRiskPremiumMaxRatio(uint256 _newRatio) external
    ↪ onlyRole(ADMIN_ROLE){ volRiskPremiumMaxRatio=_newRatio;}
function resetTrading(bool _allowTrading) external onlyRole(
    ↪ DEFAULT_ADMIN_ROLE) {allowTrading=_allowTrading;}
```

## 3.22 CVF-22

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

**Description** Should be "^0.8.0".

Listing 22:

```
2 pragma solidity 0.8.10;
```

## 3.23 CVF-23

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

**Description** It is a good practice to put external imports at the beginning or at the end of the list, but not in the middle.

Listing 23:

```
4 import "./MoretInterfaces.sol";
  import "@openzeppelin/contracts/utils/math/Math.sol";
  import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
  import "./FullMath.sol";
```

## 3.24 CVF-24

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

**Description** The type of the "_protocolDataProviderAddress" arguments should be "IProtocolDataProvider".
**Client Comment** Replaced with 'IProtocolDataProvider'.

Listing 24:

```
12 function getLendingTokenAddresses(address
   ↪ _protocolDataProviderAddress, address _tokenAddress)

17 function getTokenBalances(address _contractAddress, address
   ↪ _protocolDataProviderAddress, address _tokenAddress)
   ↪ public view returns(uint256, uint256, uint256) {

21 function getLTV(address _protocolDataProviderAddress, address
   ↪ _tokenAddress) public view returns (uint256) {

26 function getLoanTrade(address _contractAddress, address
   ↪ _protocolDataProviderAddress, int256 _aggregateDelta,
   ↪ address _underlyingAddress, bool _useVariableRate) public
   ↪ view returns(int256 _loanChange, uint256 _targetLoan,
   ↪ address _loanAddress){

33 function getCollateralTrade(address _contractAddress, address
   ↪ _protocolDataProviderAddress, uint256 _targetLoan, uint256
   ↪  _price, address _fundingAddress, address
   ↪ _underlyingAddress) public view returns(int256
   ↪ _collateralChange, address _collateralAddress) {
```

## 3.25 CVF-25

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

**Description** The semantics of the returned values is unclear.
**Recommendation** Consider giving descriptive names to the returned values and/or adding a documentation comment.
**Client Comment** Removed as no longer needed.

Listing 25:

```
13 public view returns (address, address, address){
```

## 3.26 CVF-26

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

**Description** The type of the returned values should be "IERC20".
**Client Comment** Removed as no longer needed.

Listing 26:

```
13 public view returns (address, address, address){
```

## 3.27 CVF-27

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** MarketLibrary.sol

**Description** The type of the "_tokenAddress" arguments should be "IERC20".
**Client Comment** This has to be address as address is used as argument type in contract functions of Aave.

Listing 27:

```
12 function getLendingTokenAddresses(address
      ↪ _protocolDataProviderAddress, address _tokenAddress)

17 function getTokenBalances(address _contractAddress, address
      ↪ _protocolDataProviderAddress, address _tokenAddress)
      ↪ public view returns(uint256, uint256, uint256) {

21 function getLTV(address _protocolDataProviderAddress, address
      ↪ _tokenAddress) public view returns (uint256) {

45 function balanceDef(address _tokenAddress, address
      ↪ _accountAddress) public view returns(uint256){

48 function cvtDef(uint256 _amount, address _tokenAddress) public
      ↪ view returns(uint256){

51 function cvtDecimals(uint256 _amount, address _tokenAddress)
      ↪ public view returns(uint256){

54 function cvtDecimalsInt(int256 _amount, address _tokenAddress)
      ↪ public view returns(int256 _newAmount){
```

## 3.28 CVF-28

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

**Description** The value "4" should be a named constant.
**Client Comment** Moved to a constant variable.

Listing 28:

```
24 return OptionLibrary.ToDefaultDecimals(_ltv, 4); }
```

## 3.29 CVF-29

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

**Description** Overflow is possible when converting to "int256".
**Recommendation** Consider using safe conversion.
**Client Comment** Used 'SafeCast.toInt256' instead of int256.

Listing 29:

```
31 _loanChange = int256(_targetLoan) - int256(_debtBalance);}

40 _collateralChange = int256(_requiredCollateral) - int256(
   ↪ _collateralBalance);}

43 _underlyingChange = (_aggregateDelta >=0 ? _aggregateDelta:
   ↪ int256(0)) - int256(IERC20(_underlyingAddress).balanceOf(
   ↪ _contractAddress));}

56 if(_amount > 0) _newAmount = int256(cvtDecimals(uint256(_amount)
   ↪ , _tokenAddress));
   if(_amount < 0) _newAmount = -int256(cvtDecimals(uint256(-
   ↪ _amount), _tokenAddress));}
```

## 3.30 CVF-30

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

**Description** The "decimals" property is used by UI to render token amounts in a human-friendly way. Using this property in smart contracts is discouraged. Consider treating all token amounts as integers.
**Client Comment** The reason why decimals are used are that the balances of tokens (USDC etc.), price from Oracles, as well as some parameters in Aave are all used in the contract so that the hedge trades are properly calculated with the correct integer amounts. Without decimals, the integer amounts of ETH/WBTC trades would be incorrect.

Listing 30:

```
46 return OptionLibrary.ToDefaultDecimals(ERC20(_tokenAddress).
   ↪ balanceOf(_accountAddress), ERC20(_tokenAddress).decimals
   ↪ ());}

49 return OptionLibrary.ToDefaultDecimals(_amount, ERC20(
   ↪ _tokenAddress).decimals());}

52 return OptionLibrary.ToCustomDecimals(_amount, ERC20(
   ↪ _tokenAddress).decimals());}
```

## 3.31 CVF-31

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

**Description** Should be "else if" for efficiency and readability.

Listing 31:

```
67 if(_underlyingAmt < 0 && _fundingAmt> 0){
```

## 3.32 CVF-32

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

**Description** Should be "^0.8.0".

Listing 32:

```
2 pragma solidity 0.8.10;
```

## 3.33 CVF-33

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

**Description** This interface should be moved to a separate file named "EOption.sol".

Listing 33:

```
7 interface EOption{
```

## 3.34 CVF-34

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

**Description** Events are usually named via nouns, such as "NewOption", "Exercise", "CapitalAddition" etc.

Listing 34:

```
8 event newOptionBought(address indexed _purchaser, OptionLibrary.
    ↪ Option _option, uint256 _cost, bool _inVol);
  event optionExercised(address indexed _purchaser, OptionLibrary.
    ↪ Option _option, uint256 _payoff);
10 event capitalAdded(address _recipient, uint256
    ↪ _mintMPTokenAmount, uint256 _addedValue);
  event capitalWithdrawn(address _recipient, uint256
    ↪ _burnMPTokenAmount, uint256 _withdrawValue);
  event volatilityTokenBought(address _purchaser, uint256 _time,
    ↪ uint256 _tenor, uint256 _amount, uint256 _cost);
  event volatilityTokenSold(address _seller, uint256 _time,
    ↪ uint256 _tenor, uint256 _amount, uint256 _cost);}

16 event volatilityChainBlockAdded(uint256 indexed _tenor, uint256
    ↪ _timeStamp, PriceStamp _book);
```

## 3.35 CVF-35

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

**Description** The "_recipient", "purchaser", and "_seller" parameters should be indexed.

Listing 35:

```
event capitalAdded(address _recipient, uint256
    ↪ _mintMPTokenAmount, uint256 _addedValue);
event capitalWithdrawn(address _recipient, uint256
    ↪ _burnMPTokenAmount, uint256 _withdrawValue);
event volatilityTokenBought(address _purchaser, uint256 _time,
    ↪ uint256 _tenor, uint256 _amount, uint256 _cost);
event volatilityTokenSold(address _seller, uint256 _time,
    ↪ uint256 _tenor, uint256 _amount, uint256 _cost);}
```

## 3.36 CVF-36

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

**Description** This interface should be moved to a separate file named "IVolatilityChain.sol".

Listing 36:

```
interface IVolatilityChain{
```

## 3.37 CVF-37

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** MoretInterfaces.sol

**Description** The number formats used are unclear.
**Recommendation** Consider documenting.

Listing 37:

```
17 struct PriceStamp{ uint256 startTime; uint256 endTime; uint256
   ↪ open; uint256 highest; uint256 lowest; uint256 close;
   ↪ uint256 volatility; uint256 accentus; }

20 struct VolParam{ uint256 initialVol; uint256 ltVol; uint256
   ↪ ltVolWeighted; uint256 w; uint256 p; uint256 q; }

22 function getVol(uint256 _tenor) external view returns(uint256);
   function queryPrice() external view returns(uint256, uint256);
```

## 3.38 CVF-38

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

**Description** This interface should be moved to a separate file named "IUniswapV2Router02.sol".

Listing 38:

```
27 interface IUniswapV2Router02 {
```

## 3.39 CVF-39

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

**Description** This interface should be moved to a separate file named "I1InchProtocol.sol".
**Client Comment** This interface is not used anymore so deleted.

Listing 39:

```
35 interface I1InchProtocol{
```

## 3.40 CVF-40

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

**Description** Such a long lines make the code harder to read.
**Recommendation** Consider splitting long lines.
**Client Comment** This interface is not used anymore so deleted.

Listing 40:

```
36  function getExpectedReturn( IERC20 fromToken, IERC20 destToken
      ↪ , uint256 amount, uint256 parts, uint256 flags) external
      ↪  view returns( uint256 returnAmount, uint256[] memory
      ↪ distribution);
    function getExpectedReturnWithGas(IERC20 fromToken,IERC20
      ↪ destToken,uint256 amount,uint256 parts,uint256 flags,
      ↪ uint256 destTokenEthPriceTimesGasPrice) external view
      ↪ returns(uint256 returnAmount, uint256 estimateGasAmount,
      ↪  uint256[] memory distribution);
    function getExpectedReturnWithGasMulti( IERC20[] memory tokens
      ↪ , uint256 amount, uint256[] memory parts, uint256[]
      ↪ memory flags, uint256[] memory
      ↪ destTokenEthPriceTimesGasPrices ) external view returns(
      ↪  uint256[] memory returnAmounts, uint256
      ↪ estimateGasAmount,  uint256[] memory distribution );

44  function getReserveConfigurationData(address asset) external
      ↪ view returns (uint256 decimals, uint256 ltv, uint256
      ↪ liquidationThreshold, uint256 liquidationBonus, uint256
      ↪ reserveFactor, bool usageAsCollateralEnabled, bool
      ↪ borrowingEnabled, bool stableBorrowRateEnabled, bool
      ↪ isActive, bool isFrozen);
```

## 3.41 CVF-41

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

**Description** The semantics of the returned values is unclear. Consider giving descriptive names to the returned values and/or adding documentation comments.
**Client Comment** This interface is not used anymore so deleted.

Listing 41:

```
39  function swap( IERC20 fromToken, IERC20 destToken, uint256
      ↪ amount, uint256 minReturn, uint256[] memory distribution
      ↪ , uint256 flags ) external payable returns(uint256);
40  function swapMulti( IERC20[] memory tokens, uint256 amount,
      ↪ uint256 minReturn, uint256[] memory distribution,
      ↪ uint256[] memory flags ) external payable returns(
      ↪ uint256);

53 function withdraw( address asset, uint256 amount, address to )
     ↪ external returns (uint256);

55 function repay( address asset, uint256 amount, uint256 rateMode,
     ↪  address onBehalfOf ) external returns (uint256); }
```

## 3.42 CVF-42

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

**Description** This interface should be moved to a separate file named "IProtocolDataProvider.sol".

Listing 42:

```
43 interface IProtocolDataProvider {
```

## 3.43 CVF-43

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

**Description** This interface should be moved to a separate file named "ILendingPoolAddressesProvider.sol".

Listing 43:

```
47 interface ILendingPoolAddressesProvider {
```

## 3.44 CVF-44

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

**Description** This interface should be moved to a separate file named "ILendingPool.sol".

Listing 44:

```
51 interface ILendingPool {
```

## 3.45 CVF-45

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

**Description** Should be "^0.8.0".

Listing 45:

```
2 pragma solidity 0.8.10;
```

## 3.46 CVF-46

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

**Description** These variable should be declared as "immutable".

Listing 46:

```
14 uint256 private multiplier;

17 OptionVault internal optionVault;

19 address internal underlying;
20 address internal funding;
```

## 3.47 CVF-47

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

**Description** These values could be rendered as: 0.005e18 and 0.0005e18 for readability.

Listing 47:

```
15 uint256 public settlementFee= 5 * (10 ** 15);
   uint256 public exerciseFee= 5 * (10 ** 15);

23 uint256 public swapSlippage = 5 * (10 ** 14);
```

## 3.48 CVF-48

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

**Description** The type of these variables should be "IERC20".

Listing 48:

```
19 address internal underlying;
20 address internal funding;
```

## 3.49 CVF-49

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** MoretMarketMaker.sol

**Description** There should be named constants or even enum for the valid lending pool rate modes.
**Client Comment** Leave it as it is as the impact is small due to limited usage.

Listing 49:

```
22 uint256 public lendingPoolRateMode = 2;

122   require(_newRateMode == 1 || _newRateMode == 2);
```

## 3.50 CVF-50

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

**Description** The type of the "_optionAddress" argument should be "OptionVault".

Listing 50:

```
25 constructor(string memory _name, string memory _symbol, address
   ↪ _optionAddress) ERC20(_name, _symbol){
```

## 3.51 CVF-51

- **Severity** Critical
- **Status** Fixed
- **Category** Flaw
- **Source** MoretMarketMaker.sol

**Description** This loop doesn't scale. In case there are too many expired options, this loop may not fit into the block gas limit.
**Recommendation** Consider terminating the loop if the remaining gas it below a certain threshold. Alternatively, consider accepting an additional argument specifying the maximum number of loop iterations.
**Client Comment** Added an argument '_maxContracts' to specify the max number of expiry contracts. This will also help estimate the gas by using '_maxContracts = 1'.

Listing 51:

```
38 while(_expiringId >0) {
```

## 3.52 CVF-52

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

**Description** Inside these calls the decimals number of the funding token is obtained.
**Recommendation** Consider obtaining it once in the constructor and storing in an immutable variable for future use.
**Client Comment** Changed the 'cvtDecimals' to 'toDecimals' which takes in decimals as an argument.

Listing 52:

```
42 require(_payback < MarketLibrary.balanceDef(funding, address(
     ↪ this)), "Balance insufficient.");

48   require(IERC20(funding).transfer(optionVault.getOptionHolder(
       ↪ _expiringId), MarketLibrary.cvtDecimals(_holderPayment,
       ↪ funding)), "Failed payment to holder");
     require(IERC20(funding).transfer(maintenanceAddress,
       ↪ MarketLibrary.cvtDecimals(_settleFeeAmount, funding)), "
       ↪ Failed payment to maintenance");
50   require(IERC20(funding).transfer(exerciseFeeRecipient,
       ↪ MarketLibrary.cvtDecimals(_exerciseFeeAmount, funding)),
       ↪  "Failed payment to exerciser.");}

62 uint256 _mintMPTokenAmount = MulDiv(MarketLibrary.cvtDef(
     ↪ _depositAmount, funding), multiplier, calcCapital(false,
     ↪ true));

68 uint256 _withdrawValue = MarketLibrary.cvtDecimals(MulDiv(
     ↪ calcCapital(true, true), _burnMPTokenAmount, multiplier)
     ↪ , funding);
```

## 3.53 CVF-53

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

**Description** Fixed point multiplication and division are implemented in multiple places.
**Recommendation** Consider extracting to utility functions.
**Client Comment** Changed to utility function muldiv (in FullMath.sol file)

Listing 53:

```
44  uint256 _holderPayment = _payoff == _payback? MulDiv(_payback,
      ↪  multiplier - settlementFee - exerciseFee, multiplier):
      ↪ _payback;
    uint256 _settleFeeAmount = MulDiv(_payoff, settlementFee,
      ↪ multiplier);
    uint256 _exerciseFeeAmount = MulDiv(_payoff, exerciseFee,
      ↪ multiplier);

58  if(totalSupply() > 0) _capital = MulDiv(_capital, multiplier,
      ↪ totalSupply());

62  uint256 _mintMPTokenAmount = MulDiv(MarketLibrary.cvtDef(
      ↪ _depositAmount, funding), multiplier, calcCapital(false,
      ↪ true));

68  uint256 _withdrawValue = MarketLibrary.cvtDecimals(MulDiv(
      ↪ calcCapital(true, true), _burnMPTokenAmount, multiplier)
      ↪ , funding);
```

## 3.54 CVF-54

- **Severity** Major
- **Status** Fixed
- **Category** Unclear behavior
- **Source** MoretMarketMaker.sol

**Description** Usually, "_payoff" equals "_payback" for long positions, while for short positions this is also possible. It the intention is to subtract fees only from long position payoffs, then the condition should be based on the option side rather than on the "_payoff" and "_payback" values.
**Client Comment** The equal condition is removed and replaced with the condition based on option side.

Listing 54:

```
44  uint256 _holderPayment = _payoff == _payback? MulDiv(_payback,
      ↪ multiplier - settlementFee - exerciseFee, multiplier):
      ↪ _payback;
```

## 3.55 CVF-55

- **Severity** Minor
- **Status** Info
- **Category** Documentation
- **Source** MoretMarketMaker.sol

**Description** The decimals conversion could be performed in the same "MulDiv" call that calculates the "_holderPayment", "settleFeeAmount", and "_exerciseFeeAmount" values.
**Client Comment** Still needs to run decimals conversion in the end to avoid getting the wrong payback amount

Listing 55:

```
44 uint256 _holderPayment = _payoff == _payback? MulDiv(_payback,
     ↪ multiplier - settlementFee - exerciseFee, multiplier):
     ↪ _payback;
   uint256 _settleFeeAmount = MulDiv(_payoff, settlementFee,
     ↪ multiplier);
   uint256 _exerciseFeeAmount = MulDiv(_payoff, exerciseFee,
     ↪ multiplier);

48 require(IERC20(funding).transfer(optionVault.getOptionHolder(
     ↪ _expiringId), MarketLibrary.cvtDecimals(_holderPayment,
     ↪ funding)), "Failed payment to holder");
   require(IERC20(funding).transfer(maintenanceAddress,
     ↪ MarketLibrary.cvtDecimals(_settleFeeAmount, funding)), "
     ↪ Failed payment to maintenance");
50 require(IERC20(funding).transfer(exerciseFeeRecipient,
     ↪ MarketLibrary.cvtDecimals(_exerciseFeeAmount, funding)), "
     ↪ Failed payment to exerciser.");}
```

## 3.56 CVF-56

- **Severity** Moderate
- **Status** Info
- **Category** Unclear behavior
- **Source** MoretMarketMaker.sol

**Description** In case the capital is not zero, but the total supply is zero, the capital is returned as is even when the "_average" flag is set.
**Recommendation** Consider reverting in such a case or returning "multiplier".
**Client Comment** This was intended, as when the capital is not zero but the total supply is zero, there is assumed to be 1 liquidity pool (MoretMarketMaker) token. This obviously creates arbitrage opportunity for whomever deposits tiny amount into the pool and thus can withdraw the whole lot of the capital straightaway. This arbitrage opportunity prevents LP providers from withdrawing all LP tokens.

Listing 56:

```
57 if(_average){
     if(totalSupply() > 0) _capital = MulDiv(_capital , multiplier
       ↪ , totalSupply());
     if(totalSupply() == 0 && _capital == 0) _capital = multiplier
       ↪ ;}}
```

## 3.57 CVF-57

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

**Description** The decimals conversion could be merged with the "MulDiv" call.
**Client Comment** Decided to keep it as it is as the solution might be more complicated.

Listing 57:

```
62 uint256 _mintMPTokenAmount = MulDiv(MarketLibrary.cvtDef(
     ↪ _depositAmount, funding), multiplier, calcCapital(false,
     ↪ true));

68 uint256 _withdrawValue = MarketLibrary.cvtDecimals(MulDiv(
     ↪ calcCapital(true, true), _burnMPTokenAmount , multiplier)
     ↪ , funding);
```

## 3.58 CVF-58

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

**Description** This event is redundant, as the preceding "transferFrom" and "_mint" calls emit events that contain enough information about the operation.
**Client Comment** Removed this event.

Listing 58:

```
65 emit capitalAdded(msg.sender, _depositAmount, _mintMPTokenAmount
   ↪ );}
```

## 3.59 CVF-59

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

**Description** This check is redundant, as the "transfer" call will anyway fail on insufficient balance.
**Client Comment** Removed.

Listing 59:

```
69 require(IERC20(funding).balanceOf(address(this)) >
   ↪ _withdrawValue,"Insufficient balance");
```

## 3.60 CVF-60

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

**Description** This event is redundant, as the preceding "_burn" and "transfer" calls emit events that contain enough information about the operation.
**Client Comment** Removed this event.

Listing 60:

```
72 emit capitalWithdrawn(msg.sender, _burnMPTokenAmount,
   ↪ _withdrawValue);}
```

## 3.61 CVF-61

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** MoretMarketMaker.sol

**Description** The returned value is ignored.
**Client Comment** Added require clause to capture approve results.

Listing 61:

```
75  ERC20(_tokenAddress).approve(_spenderAddress, _amount);}

83  ERC20(_fromAddress).approve(_router, _fromAmt);

90  ERC20(_fromAddress).approve(_aggregator, _fromAmt);

109   ERC20(_loanAddress).approve(_lendingPoolAddress, uint256(-
        ↪ _loanTradeAmount));
110   ERC20(underlying).approve(_lendingPoolAddress, uint256(-
        ↪ _loanTradeAmount));
```

## 3.62 CVF-62

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

**Description** Should be "else" instead of "if (...)".
**Client Comment** The whole block has been deleted as no longer used.

Listing 62:

```
80  if(!_useAggregator) swapByRouter(_fromAddress, _toAddress,
      ↪ _router, _fromAmt, _toAmt, _parameter);}
```

## 3.63 CVF-63

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

**Description** The type of the "_fromAddress" and "_toAddress" should be "IERC20".
**Client Comment** Both functions are redundant and deleted.

Listing 63:

```
82 function swapByRouter(address _fromAddress, address _toAddress,
   ↪ address _router, uint256 _fromAmt, uint256 _toAmt, uint256
   ↪  _deadline) internal {

89 function swapByAggregator(address _fromAddress, address
   ↪ _toAddress, address _aggregator, uint256 _fromAmt, uint256
   ↪  _toAmt, uint256 _parts) internal {
```

## 3.64 CVF-64

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

**Description** The type of the "_router" argument should be "IUniswapV2Router02".

Listing 64:

```
82 function swapByRouter(address _fromAddress, address _toAddress,
   ↪ address _router, uint256 _fromAmt, uint256 _toAmt, uint256
   ↪  _deadline) internal {
```

## 3.65 CVF-65

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

**Description** The type of the "_aggregator" argument should be "I1InchProtocol".
**Client Comment** This function has been remodeled so this argument is not longer used.

Listing 65:

```
89 function swapByAggregator(address _fromAddress, address
   ↪ _toAddress, address _aggregator, uint256 _fromAmt, uint256
   ↪  _toAmt, uint256 _parts) internal {
```

## 3.66 CVF-66

- **Severity** Major
- **Status** Fixed
- **Category** Procedural
- **Source** MoretMarketMaker.sol

**Description** The "increaseAllowance" function is not a standard ERC-20 function.
**Recommendation** Consider using the combination of "allowance' and "approve" calls instead.
**Client Comment** Changed to approve and added required clause (related to #62).

Listing 66:

```
101 ERC20(funding).increaseAllowance(_lendingPoolAddress, uint256(
    ↪ _collateralChange));

114 ERC20(_collateralAddress).increaseAllowance(_lendingPoolAddress,
    ↪  uint256(-_collateralChange));
```

## 3.67 CVF-67

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

**Description** The expression "uint256(-loanTradeAmount)" is calculated several times.
**Recommendation** Consider calculating once and reusing.

Listing 67:

```
108 require(ERC20(underlying).balanceOf(address(this))>= uint256(-
    ↪ _loanTradeAmount), "not enough token to repay loans");
ERC20(_loanAddress).approve(_lendingPoolAddress, uint256(-
    ↪ _loanTradeAmount));
110 ERC20(underlying).approve(_lendingPoolAddress, uint256(-
    ↪ _loanTradeAmount));
ILendingPool(_lendingPoolAddress).repay(underlying, uint256(-
    ↪ _loanTradeAmount), lendingPoolRateMode, address(this));}
```

## 3.68 CVF-68

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

**Description** These functions should emit some events.
**Client Comment** Added events 'ResetParameter'.

Listing 68:

```
117 function resetSettlementFee(uint256 _newFee) external onlyRole(
    ↪ ADMIN_ROLE){ require(_newFee < multiplier); settlementFee
    ↪ = _newFee;}
    function resetExerciseFee(uint256 _newFee) external onlyRole(
    ↪ ADMIN_ROLE){ require(_newFee < multiplier); exerciseFee =
    ↪ _newFee;}
    function resetMaintenance(address _newAddress) external onlyRole
    ↪ (ADMIN_ROLE){ maintenanceAddress = _newAddress;}
120 function resetSlippage(uint256 _slippage) external onlyRole(
    ↪ ADMIN_ROLE){ swapSlippage = _slippage;}
    function resetLendingPoolRateMode(uint256 _newRateMode) external
    ↪  onlyRole(ADMIN_ROLE) {
```

## 3.69 CVF-69

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

**Description** Should be "^0.8.0".

Listing 69:

```
7 pragma solidity 0.8.10;
```

## 3.70 CVF-70

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

**Description** We didn't review this file.

Listing 70:

```
9 import "./FullMath.sol";
```

## 3.71 CVF-71

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

**Description** Such a long line make the code harder to read.
**Recommendation** Consider putting each field at a separate line.

Listing 71:

```
17 struct Option { PayoffType poType; OptionSide side; OptionStatus
   ↪  status; address holder; uint256 id; uint256 createTime;
   ↪ uint256 effectiveTime; uint256 tenor; uint256 maturity;
   ↪ uint256 exerciseTime; uint256 amount;uint256 spot;uint256
   ↪ strike;uint256 volatility;uint256 premium;uint256 cost;}
```

## 3.72 CVF-72

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

**Description** This struct looks more like a simple fraction rather than a percent. Consider renaming.
**Client Comment** Removed as no longer used.

Listing 72:

```
18 struct Percent{ uint256 numerator; uint256 denominator;}
```

## 3.73 CVF-73

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

**Description** Constant are usually named IN_UPPER_CASE.
**Client Comment** Renamed as BASE and DECIMALS.

Listing 73:

```
19 uint256 public constant DefaultMultiplier = 10 ** 18;
20 uint256 public constant DefaultDecimals = 18;
```

## 3.74 CVF-74

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

**Description** This value could be rendered as "1e18".

Listing 74:

```
19 uint256 public constant DefaultMultiplier = 10 ** 18;
```

## 3.75 CVF-75

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** OptionLibrary.sol

**Description** Consider deriving the default multiplier from the default decimals to guarantee consistency between them.
**Client Comment** Leave it as it is.

Listing 75:

```
19 uint256 public constant DefaultMultiplier = 10 ** 18;
```

## 3.76 CVF-76

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

**Description** These functions are redundant as they return constants that are public by themselves.
**Client Comment** Removed.

Listing 76:

```
22 function Multiplier() public pure returns (uint256){return
   ↪ DefaultMultiplier;}
function Decimals() public pure returns (uint256) {return
   ↪ DefaultDecimals;}
```

## 3.77 CVF-77

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

**Description** This constant is used directly in many calculations.
**Recommendation** Consider implementing utility functions for fixed–point arithmetic to make the code simpler and easier to read.
**Client Comment** Moved to FullMath as part of utility.

Listing 77:

```
19 uint256 public constant DefaultMultiplier = 10 ** 18;
```

## 3.78 CVF-78

- **Severity** Major
- **Status** Fixed
- **Category** Flaw
- **Source** OptionLibrary.sol

**Description** This function cannot handle numbers with more than 18 decimals.
**Recommendation** Consider implementing support for such ability.
**Client Comment** Added clauses to adjust decimals if the number is more than 18 decimals.

Listing 78:

```
24 function ToDefaultDecimals(uint256 _rawData, uint256
    ↪ _rawDataDecimals) public pure returns(uint256){

27 function ToCustomDecimals(uint256 _rawData, uint256
    ↪ _rawDataDecimals) public pure returns(uint256){
```

## 3.79 CVF-79

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

**Description** This calculation could be simplified in a quite common case when "_rawDataDecimals" equals to "DefaultDecimals".
**Client Comment** The default case is defined first before the if clauses are executed.

Listing 79:

```
26 return _rawData * (10** (DefaultDecimals - _rawDataDecimals));}

29 return _rawData / (10** (DefaultDecimals - _rawDataDecimals));}
```

## 3.80 CVF-80

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

**Description** For consistency with the argument names, a better name would be "ToRawDecimals".
**Client Comment** Moved to MarketLibrary and renamed 'toDecimals'.

Listing 80:

```
27 function ToCustomDecimals(uint256 _rawData, uint256
     ↪ _rawDataDecimals) public pure returns(uint256){
```

## 3.81 CVF-81

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

**Description** The "MulDiv" function is very efficient in general case, while in specific cases more efficient approaches do exists. For example, when the denominator is known at compiler time, its modular inversion could be precomputed. Also, when the denominator is less than 2^128, the approach described here could be used: https://medium.com/coinmonks/math-in-solidity-part-3-percents-and-proportions-4db014e080b1#4821
**Client Comment** Interesting approach. A new function is created in FullMath.ethmul. However, after testing it, there seem to be unexpected behaviour such as very large result. Therefore this approach is not used here.

Listing 81:

```
35    return MulDiv(_intrinsicValue, _amount, DefaultMultiplier); }

43  return MulDiv(_amount * 4, MulDiv(_b, _volatility, _a -
      ↪ _moneyness), 10 * DefaultMultiplier);}

47    uint256 _timeValue = MulDiv(calcTimeValue(_strike, _price,
        ↪ _volatility, _amount), _price, DefaultMultiplier);

53  if(_side == OptionSide.Sell && _poType == PayoffType.Put){ _cost
      ↪  = MulDiv(_amount, _strike, DefaultMultiplier) - _cost;}
    if(_side == OptionSide.Sell && _poType == PayoffType.Call){
      ↪ _cost = MulDiv(_amount, _price, DefaultMultiplier) - _cost
      ↪ ;}}

60  return MulDiv(_option.amount, _price, DefaultMultiplier);}

80  if(_adjustUpward) _adjustedAmount = MulDiv(_amount,
      ↪ DefaultMultiplier + _slippage + _loanInterest,
      ↪ DefaultMultiplier);
```

## 3.82 CVF-82

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

**Description** This function is overcomplicated. It could be simplified as: uint v = DefaultMultiplier < _volatility ? DefaultMultiplier : _volatility; uint m = _strike > _price ? MulDiv (_price, DefaultMultiplier, _strike) : MulDiv (_strike, DefaultMultiplier, _price); return MulDiv (_amount, MulDiv (v, _volatility, 5 * (DefaultMultiplier + v / 2 - m), DefaultMultiplier);
**Client Comment** This is further improved to base the calculation on atm'_premium'.

Listing 82:

```
37 function calcTimeValue(uint256 _strike, uint256 _price, uint256
   ↪ _volatility, uint256 _amount) private pure returns (
   ↪ uint256){
```

## 3.83 CVF-83

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

**Description** "return 0" would be more clear.

Listing 83:

```
38 if(_volatility == 0) return _volatility;
```

## 3.84 CVF-84

- **Severity** Minor
- **Status** Fixed
- **Category** Overflow/Underflow
- **Source** OptionLibrary.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 overflow. Consider using the "MulDiv" function.

Listing 84:

```
39 uint256 _moneyness = _strike > _price? (_price *
   ↪ DefaultMultiplier / _strike) : (_strike *
   ↪ DefaultMultiplier / _price); // always in (0,1]
```

## 3.85 CVF-85

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

**Description** This variable is redundant. Just calculate "_b" as "(_volatility > DefaultMultiplier ? DefaultMultiplier, _volatility) / 2" and "_a" as "DefaultMultiplier + _b".

Listing 85:

```
40 uint256 _midPoint = DefaultMultiplier - (_volatility >
   ↪ DefaultMultiplier? DefaultMultiplier: _volatility) / 2 ;
   ↪ // always in [0.5, 1]
```

## 3.86 CVF-86

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

**Description** The same name "_moneyness" is used for different values. This makes the code harder to read.
**Recommendation** Consider using different names.

Listing 86:

```
39 uint256 _moneyness = _strike > _price? (_price *
   ↪ DefaultMultiplier / _strike) : (_strike *
   ↪ DefaultMultiplier / _price); // always in (0,1]

63 uint256 _moneyness = MulDiv(_price, DefaultMultiplier, _strike);

68 uint256 _moneyness = MulDiv(_price, DefaultMultiplier, _strike);
```

## 3.87 CVF-87

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** OptionLibrary.sol

**Description** This will revert when _a==_moneyness but volatility!=0 (e.g. it equals 1). Consider returning 0 in this case.
**Client Comment** _a' by definition is larger than '_m'.

Listing 87:

```
43 return MulDiv(_amount * 4, MulDiv(_b, _volatility, _a -
   ↪ _moneyness), 10 * DefaultMultiplier);}
```

## 3.88 CVF-88

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

**Description** This "MulDiv" call should be moved into the "calcTimeValue" function.

Listing 88:

```
47 uint256 _timeValue = MulDiv(calcTimeValue(_strike, _price,
   ↪ _volatility, _amount), _price, DefaultMultiplier);
```

## 3.89 CVF-89

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

**Description** This code could be simplified as: if _side == OptionSide.Sell) { if (_poType == PayoffType.Put) { .. } else if (_poType == PayoffType.Call) { ... } }

Listing 89:

```
53 if(_side == OptionSide.Sell && _poType == PayoffType.Put){ _cost
   ↪ = MulDiv(_amount, _strike, DefaultMultiplier) - _cost;}
   if(_side == OptionSide.Sell && _poType == PayoffType.Call){
   ↪ _cost = MulDiv(_amount, _price, DefaultMultiplier) - _cost
   ↪ ;}}
```

## 3.90 CVF-90

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

**Description** The conversion into "int256" may overflow.
**Recommendation** Consider using safe conversion.
**Client Comment** int256() has been replaced with 'SafeCast.toInt256()' to raise exception at overflow events. The same change has also been applied on other contracts.

Listing 90:

```
64 int256 _d = int256(MulDiv(2 * DefaultMultiplier, (_moneyness *
   ↪ DefaultMultiplier).sqrt(), _vol)) - int256(MulDiv(2 *
   ↪ DefaultMultiplier, DefaultMultiplier, _vol)) + int256(
   ↪ _vol/ 2);

69 int256 _d = int256(MulDiv(2 * DefaultMultiplier, (_moneyness *
   ↪ DefaultMultiplier).sqrt(), _vol)) - int256(MulDiv(2 *
   ↪ DefaultMultiplier, DefaultMultiplier, _vol)) + int256(
   ↪ _vol/ 2);
```

## 3.91 CVF-91

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

**Description** The expression "MulDiv(2 * DefaultMultiplier, DefaultMultiplier, _vol)" is equivalent to "2 * DefaultMultiplier * DefaultMultiplier / _vol"

Listing 91:

```
64 int256 _d = int256(MulDiv(2 * DefaultMultiplier, (_moneyness *
   ↪ DefaultMultiplier).sqrt(), _vol)) - int256(MulDiv(2 *
   ↪ DefaultMultiplier, DefaultMultiplier, _vol)) + int256(
   ↪ _vol/ 2);

69 int256 _d = int256(MulDiv(2 * DefaultMultiplier, (_moneyness *
   ↪ DefaultMultiplier).sqrt(), _vol)) - int256(MulDiv(2 *
   ↪ DefaultMultiplier, DefaultMultiplier, _vol)) + int256(
   ↪ _vol/ 2);
```

## 3.92 CVF-92

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

**Description** This expression is coded twice.
**Recommendation** Consider extracting into a utility function.
**Client Comment** Leave it as it is.

Listing 92:

```
64 int256 _d = int256(MulDiv(2 * DefaultMultiplier, (_moneyness *
   ↪ DefaultMultiplier).sqrt(), _vol)) - int256(MulDiv(2 *
   ↪ DefaultMultiplier, DefaultMultiplier, _vol)) + int256(
   ↪ _vol/ 2);

69 int256 _d = int256(MulDiv(2 * DefaultMultiplier, (_moneyness *
   ↪ DefaultMultiplier).sqrt(), _vol)) - int256(MulDiv(2 *
   ↪ DefaultMultiplier, DefaultMultiplier, _vol)) + int256(
   ↪ _vol/ 2);
```

## 3.93 CVF-93

- **Severity** Major
- **Status** Fixed
- **Category** Readability
- **Source** OptionLibrary.sol

**Description** This line should be moved to the "else" branch of the conditional statement below.
**Client Comment** This logic has been deleted and no _adjustedStrike is now used.

Listing 93:

```
73 _adjustedStrike = adjustSlippage(_strike, false, _slippage, 0);
   ↪ // downward
```

## 3.94 CVF-94

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

**Description** The expression "DefaultMultiplier + _slippage + _loanInterest" is coded twice.
**Recommendation** Consider calculating once before the conditional statement.
**Client Comment** Leave it as it is.

Listing 94:

```
80 if(_adjustUpward) _adjustedAmount = MulDiv(_amount,
   ↪ DefaultMultiplier + _slippage + _loanInterest,
   ↪ DefaultMultiplier);
if(!_adjustUpward) _adjustedAmount = MulDiv(_amount,
   ↪ DefaultMultiplier, DefaultMultiplier + _slippage+
   ↪ _loanInterest);}
```

## 3.95 CVF-95

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

**Description** Should be "else" instead of "if (...)".

Listing 95:

```
81 if(!_adjustUpward) _adjustedAmount = MulDiv(_amount,
   ↪ DefaultMultiplier, DefaultMultiplier + _slippage+
   ↪ _loanInterest);}
```

## 3.96 CVF-96

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

**Description** Over- and underflow possible when converting types. Consider using safe conversions.
**Client Comment** Used SafeCast.toInt256.

Listing 96:

```
84 _oppositeAmount = int256(MulDiv(adjustSlippage(_amount >=0 ?
   ↪ uint256(_amount): uint256(-_amount), _amount>0, _slippage,
   ↪  0), _price, DefaultMultiplier)) * (_amount >=0? int256
   ↪ (-1): int256(1));}
```

## 3.97 CVF-97

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

**Description** Should be "^0.8.0".

Listing 97:

```
7 pragma solidity 0.8.10;
```

## 3.98 CVF-98

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

**Description** These two variables could be merged into a single variable of type "OptionLibrary.Option []".

Listing 98:

```
18 mapping(uint256=> OptionLibrary.Option) internal optionsList;
   uint256 public optionCounter = 0;
```

## 3.99 CVF-99

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

**Description** These variables should be declared as immutable.

Listing 99:

```
22 IVolatilityChain internal volatilityChain;

24 address public volChainAddress;
   address public aaveAddress;
   address public underlying;
   address public funding;
```

## 3.100 CVF-100

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

**Description** The type of this variable should be "IVolatilityChain".

Listing 100:

```
24 address public volChainAddress;
```

## 3.101 CVF-101

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

**Description** The type of this variable should be "ILendingPoolAddressesProvider".

Listing 101:

```
25 address public aaveAddress;
```

## 3.102 CVF-102

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

**Description** The type of these variables should be "IERC20".
**Client Comment** Used ERC20 for decimals.

Listing 102:

```
26 address public underlying;
   address public funding;
```

## 3.103 CVF-103

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

**Description** The types of the arguments should be: "IVolatilityChain", "IERC20", "IERC20", "ILendingPoolAddressesProvider" respectively.

Listing 103:

```
29 constructor( address _volChainAddress, address _underlying,
     ↪ address _funding, address _aaveAddress){
```

## 3.104 CVF-104

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

**Description** The "abi.encodePacked" call is redundant. Just do: keccak256(bytes(volatilityChain.getDecription()))
**Client Comment** Deleted.

Listing 104:

```
38 function descriptionHash() external view returns (bytes32) {
   ↪ return keccak256(abi.encodePacked(volatilityChain.
   ↪ getDecription()));}
```

## 3.105 CVF-105

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** OptionVault.sol

**Description** There are no range checks for arguments. Consider adding proper checks.
**Client Comment** Added check for tenor.

Listing 105:

```
40 function addOption(uint256 _tenor, uint256 _strike, uint256
   ↪ _amount, OptionLibrary.PayoffType _poType, OptionLibrary.
   ↪ OptionSide _side, uint256 _premium, uint256 _cost, uint256
   ↪  _price, uint256 _volatility, address _holder) external
   ↪ onlyRole(EXCHANGE_ROLE) returns(uint256 _id) {
```

## 3.106 CVF-106

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** OptionVault.sol

**Description** Consider adding comments to the zero values with the argument names.

Listing 106:

```
43 optionsList[_id] = OptionLibrary.Option(_poType, _side,
   ↪ OptionLibrary.OptionStatus.Draft, _holder, _id, block.
   ↪ timestamp, 0, _tenor, 0, 0, _amount, _price, _strike,
   ↪ _volatility, _premium, _cost);}
```

## 3.107 CVF-107

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** OptionVault.sol

**Description** This function silently returns zero address for invalid option ID. Consider reverting in such a case.
**Client Comment** Deleted.

Listing 107:

```
45 function getOptionHolder(uint256 _id) external view returns(
     ↪ address) { return optionsList[_id].holder;}
```

## 3.108 CVF-108

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** OptionVault.sol

**Description** This function silently returns empty option for invalid option ID.
**Recommendation** Consider reverting in such a case.
**Client Comment** Added require clause.

Listing 108:

```
48 function getOption(uint256 _id) external view returns(
     ↪ OptionLibrary.Option memory) {return optionsList[_id];}
```

## 3.109 CVF-109

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

**Description** This loop doesn't scale. In case of too many active options, it could exceed the block gas limit.
**Recommendation** Consider maintaining the aggregating notional value in the storage and updating it when options are created or expired.
**Client Comment** This function is deleted as is obsolete.

Listing 109:

```
52 for(uint256 i=0;i<activeOptions.length();i++){
     _notional += optionsList[uint256(activeOptions.at(i))].amount
       ↪ ;} }
```

## 3.110 CVF-110

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

**Description** The expression "activeOptions.length()" is calculated on every loop iteration.
**Recommendation** Consider calculating once before the loop.
**Client Comment** Replaced with 'activeContractCount'.

Listing 110:

```
52  for(uint256 i=0;i<activeOptions.length();i++){
70  for(uint256 i=0;i<activeOptions.length();i++){
96  for(uint256 i=0;i<activeOptions.length();i++){
109 for(uint256 i=0;i<activeOptions.length();i++){
123 for(uint256 i=0;i<activeOptions.length();i++){
132 for(uint256 i=0;i<activeOptions.length();i++){
```

## 3.111 CVF-111

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** OptionVault.sol

**Description** This function is not able to return negative capital.
**Recommendation** Consider implementing such ability.
**Client Comment** Negative capital is not allowed conceptually. The min would be zero.

Listing 111:

```
55 function getGrossCapital(address _address) external view returns
   ↪ (uint256 _capital){
```

## 3.112 CVF-112

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

**Description** The :"MulDiv" function is efficient in general case but in certain special cases more efficient approaches do exists. For example, when the denominator is known at compile time, its modular inverse could be precomputed. Also, when the denominator is less than 2^128, the approach described here could be used: https://medium.com/coinmonks/math-in-solidity-part-3-percents-and-proportions-4db014e080b1#4821
**Client Comment** ethmul' and 'ethdiv' are implemented using the method you prescribed.

Listing 112:

```
58  _capital = _funding_balance + _collateral_balance + MulDiv(
      ↪ _underlying_balance, _price, OptionLibrary.Multiplier());
    require(_capital > MulDiv(_debt_balance, _price, OptionLibrary.
      ↪ Multiplier()), "Negative equity.");
60  _capital -= MulDiv(_debt_balance, _price, OptionLibrary.
      ↪ Multiplier());}

66  return MulDiv(Math.max(_deltaZero>=0?uint256(_deltaZero):uint256
      ↪ (-_deltaZero), _deltaMax>=0?uint256(_deltaMax):uint256(-
      ↪ _deltaMax) ), _price, OptionLibrary.Multiplier());}

89      _delta = int256(MulDiv(OptionLibrary.calcDelta(_price,
          ↪ optionsList[_id].strike, _vol), optionsList[_id].
          ↪ amount, OptionLibrary.Multiplier() ));

103   _gamma = int256(MulDiv(OptionLibrary.calcGamma(_price,
        ↪ optionsList[_id].strike, _vol), optionsList[_id].amount,
        ↪  OptionLibrary.Multiplier() ));

178   _repaySwapValue = MarketLibrary.cvtDecimals(MulDiv(
        ↪ _repayAmount, _price, OptionLibrary.Multiplier()),
        ↪ funding);
```

## 3.113 CVF-113

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

**Description** The expression "MulDiv(_debt_balance, _price, OptionLibrary.Multiplier())" is calculated twice. Consider calculating once and reusing.

Listing 113:

```
59 require(_capital > MulDiv(_debt_balance, _price, OptionLibrary.
   ↪ Multiplier()), "Negative equity.");
60 _capital -= MulDiv(_debt_balance, _price, OptionLibrary.
   ↪ Multiplier());}
```

## 3.114 CVF-114

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

**Description** Fixed-point multiplication is implemented in many places. Consider moving to a utility function.

Listing 114:

```
58 _capital = _funding_balance + _collateral_balance + MulDiv(
     ↪ _underlying_balance, _price, OptionLibrary.Multiplier());
   require(_capital > MulDiv(_debt_balance, _price, OptionLibrary.
     ↪ Multiplier()), "Negative equity.");
60 _capital -= MulDiv(_debt_balance, _price, OptionLibrary.
     ↪ Multiplier());}

66 return MulDiv(Math.max(_deltaZero>=0?uint256(_deltaZero):uint256
     ↪ (-_deltaZero), _deltaMax>=0?uint256(_deltaMax):uint256(-
     ↪ _deltaMax) ), _price, OptionLibrary.Multiplier());}

89       _delta = int256(MulDiv(OptionLibrary.calcDelta(_price,
           ↪ optionsList[_id].strike, _vol), optionsList[_id].
           ↪ amount, OptionLibrary.Multiplier() ));

103    _gamma = int256(MulDiv(OptionLibrary.calcGamma(_price,
         ↪ optionsList[_id].strike, _vol), optionsList[_id].amount,
         ↪  OptionLibrary.Multiplier() ));

114 _gamma = int256(OptionLibrary.calcGamma(OptionLibrary.Multiplier
     ↪ (), OptionLibrary.Multiplier(), _vol));}

178    _repaySwapValue = MarketLibrary.cvtDecimals(MulDiv(
         ↪ _repayAmount, _price, OptionLibrary.Multiplier()),
         ↪ funding);
```

## 3.115 CVF-115

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

**Description** The "1e5" value should be a named constant.
**Client Comment** Replaced with constant variable SCALING.

Listing 115:

```
64 int256 _deltaZero = calculateAggregateDelta(_price / 1e5, true);
   int256 _deltaMax = calculateAggregateDelta(_price * 1e5, true);
```

## 3.116 CVF-116

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

**Description** Consider implementing and using a generic "abs" function, to make the code simpler and easier to read.

Listing 116:

```
66 return MulDiv(Math.max(_deltaZero>=0?uint256(_deltaZero):uint256
   ↪ (-_deltaZero), _deltaMax>=0?uint256(_deltaMax):uint256(-
   ↪ _deltaMax) ), _price, OptionLibrary.Multiplier());}
```

## 3.117 CVF-117

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

**Description** This loop doesn't scale. Consider maintaining the sell PUT collateral variable in the storage and updating it when an option is created or expired.
**Client Comment** Saved to variable 'sellPutCollaterals'.

Listing 117:

```
70 for(uint256 i=0;i<activeOptions.length();i++){
```

## 3.118 CVF-118

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

**Description** The "optionsList[_id].status == OptionLibrary.OptionStatus.Active" part is redundant as the loop is iterating over active options only.
**Client Comment** Function is removed.

Listing 118:

```
72 if(optionsList[_id].status== OptionLibrary.OptionStatus.Active
   ↪ && optionsList[_id].side == OptionLibrary.OptionSide.Sell
   ↪ && optionsList[_id].poType==OptionLibrary.PayoffType.Put){
```

## 3.119 CVF-119

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

**Description** The expression "optionsList[_id]" is calculated several times. Consider calculating once and reusing.
**Client Comment** Function is removed.

Listing 119:

```
72 if(optionsList[_id].status== OptionLibrary.OptionStatus.Active
     ↪ && optionsList[_id].side == OptionLibrary.OptionSide.Sell
     ↪ && optionsList[_id].poType==OptionLibrary.PayoffType.Put){
_collateral += optionsList[_id].calcNotionalExposure(
     ↪ optionsList[_id].strike);}}}
```

## 3.120 CVF-120

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

**Description** For a short position, the payback value returned by this function is min(strike, price) * amount regardless of the option type. This fact allows significantly simplifying the function.
**Client Comment** The logic has been updated.

Listing 120:

```
75 function getContractPayoff(uint256 _id) external view returns(
     ↪ uint256 _payoff, uint256 _payback){
```

## 3.121 CVF-121

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

**Description** The expression "optionsList[_id]" is calculated several times.
**Recommendation** Consider calculating once and reusing.
**Client Comment** Moved to 'OptionLibrary.calcPayoff'.

Listing 121:

```
77  _payoff = optionsList[_id].calcPayoff(_price);

79  if(optionsList[_id].side == OptionLibrary.OptionSide.Sell &&
      ↪ optionsList[_id].poType==OptionLibrary.PayoffType.Call){
80    _payback = optionsList[_id].calcNotionalExposure(_price) -
        ↪ _payoff;}
    if(optionsList[_id].side == OptionLibrary.OptionSide.Sell &&
      ↪ optionsList[_id].poType==OptionLibrary.PayoffType.Put){
      _payback = optionsList[_id].calcNotionalExposure(optionsList[
        ↪ _id].strike) - _payoff;}}
```

## 3.122 CVF-122

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

**Description** This could be simplified as: if (optionsList[_id].side == OptionLibrary.OptionSide.Sell) { if (optionsList[_id].poType == OptionLibrary.PayoffType.Call) {...} else if (optionsList[_id].poType == OptionLibrary.PayoffType.Put) {...} }
**Client Comment** Fixed in 'OptionLibrary.calcPayoff'.

Listing 122:

```
79  if(optionsList[_id].side == OptionLibrary.OptionSide.Sell &&
      ↪ optionsList[_id].poType==OptionLibrary.PayoffType.Call){

81  if(optionsList[_id].side == OptionLibrary.OptionSide.Sell &&
      ↪ optionsList[_id].poType==OptionLibrary.PayoffType.Put){
```

## 3.123 CVF-123

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

**Description** Should be "else if" for readability.

Listing 123:

```
81 if(optionsList[_id].side == OptionLibrary.OptionSide.Sell &&
   ↪ optionsList[_id].poType==OptionLibrary.PayoffType.Put){
```

## 3.124 CVF-124

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

**Description** This function looks cumbersome with three levels of nested conditional statements.
**Recommendation** Consider refactoring.
**Client Comment** Moved to 'OptionLibrary.calcDelta'.

Listing 124:

```
84 function calculateContractDelta(uint256 _id, uint256 _price,
   ↪ bool _includeExpiring) public view returns(int256 _delta){
```

## 3.125 CVF-125

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

**Description** The expression "optionsList[_id]" is calculated several times.
**Recommendation** Consider calculating once and reusing.

Listing 125:

```
86  if(optionsList[_id].status== OptionLibrary.OptionStatus.Active
      ↪ && (_includeExpiring || (optionsList[_id].maturity > block
      ↪ .timestamp))){
    if(optionsList[_id].side==OptionLibrary.OptionSide.Buy){
      uint256 _vol = volatilityChain.getVol(optionsList[_id].
        ↪ maturity - Math.min(optionsList[_id].maturity, block.
        ↪ timestamp));
      _delta = int256(MulDiv(OptionLibrary.calcDelta(_price,
        ↪ optionsList[_id].strike, _vol), optionsList[_id].
        ↪ amount, OptionLibrary.Multiplier() ));
90    if(optionsList[_id].poType==OptionLibrary.PayoffType.Put) {
        ↪ _delta = -int256(optionsList[_id].amount) + _delta; }}
    if(optionsList[_id].side==OptionLibrary.OptionSide.Sell &&
      ↪ optionsList[_id].poType==OptionLibrary.PayoffType.Call){
      _delta = int256(optionsList[_id].amount);}}} // collateral
        ↪ for sell call options. zero for sell put options
```

## 3.126 CVF-126

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

**Description** Overflow is possible when converting to "int256".
**Recommendation** Consider using safe conversion.

Listing 126:

```
89      _delta = int256(MulDiv(OptionLibrary.calcDelta(_price,
          ↪ optionsList[_id].strike, _vol), optionsList[_id].
          ↪ amount, OptionLibrary.Multiplier()));
90      if(optionsList[_id].poType==OptionLibrary.PayoffType.Put) {
          ↪ _delta = -int256(optionsList[_id].amount) + _delta; }}

92      _delta = int256(optionsList[_id].amount);}}} // collateral
          ↪ for sell call options. zero for sell put options

103   _gamma = int256(MulDiv(OptionLibrary.calcGamma(_price,
        ↪ optionsList[_id].strike, _vol), optionsList[_id].amount,
        ↪  OptionLibrary.Multiplier()));

114 _gamma = int256(OptionLibrary.calcGamma(OptionLibrary.Multiplier
      ↪ (), OptionLibrary.Multiplier(), _vol));}

159 _tradeUnderlyingAmount = (_aggregateDelta >= 0? _aggregateDelta:
      ↪  int256(0)) - int256(MarketLibrary.balanceDef(underlying,
      ↪ _address));
```

## 3.127 CVF-127

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

**Description** This could be simplified as: if (...) _delta -= int256(optionsList[_id].amount);

Listing 127:

```
90 if(optionsList[_id].poType==OptionLibrary.PayoffType.Put) {
     ↪ _delta = -int256(optionsList[_id].amount) + _delta; }}
```

## 3.128 CVF-128

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

**Description** Should be "else if" for readability.

Listing 128:

```
91 if(optionsList[_id].side==OptionLibrary.OptionSide.Sell &&
   ↪ optionsList[_id].poType==OptionLibrary.PayoffType.Call){
```

## 3.129 CVF-129

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

**Description** Theseloops don't scale. In case of too many active options they could exceed the block gas limit.
**Recommendation** Consider refactoring.
**Client Comment** Related functions are supposed to be used in view only.

Listing 129:

```
96  for(uint256 i=0;i<activeOptions.length();i++){
109 for(uint256 i=0;i<activeOptions.length();i++){
```

## 3.130 CVF-130

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

**Description** This commented out code should be removed.

Listing 130:

```
116 // function validateOption(uint256 _id, address _holder)
    ↪ external view {
//    require(optionsList[_id].holder== _holder, "Not the owner
    ↪ .");
//    require(optionsList[_id].maturity >= block.timestamp, "
    ↪ Option has expired.");
//    require(optionsList[_id].status==OptionLibrary.OptionStatus
    ↪ .Active, "Not active option.");}

145 // function stampExercisedOption(uint256 _id) external onlyRole(
    ↪ EXCHANGE_ROLE){
//      optionsList[_id].exerciseTime = block.timestamp;
//      optionsList[_id].status = OptionLibrary.OptionStatus.
    ↪ Exercised;}
```

## 3.131 CVF-131

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

**Description** This function is redundant, as "getExpiringOptionId" function could be used instead.
**Client Comment** Keep it as it is. No additional gas is consumed as it is view only.

Listing 131:

```
121 function anyOptionExpiring() external view returns(bool
    ↪ _isExpiring) {
```

## 3.132 CVF-132

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

**Description** This loop doesn't scale. In case active options would be sorted or grouped by maturity, it would be easy to tell whether any of them is expiring.
**Client Comment** It would be alternative way to group by maturity. However, the available maturities are infinite, making grouping impossible.

Listing 132:

```
123 for(uint256 i=0;i<activeOptions.length();i++){
```

## 3.133 CVF-133

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

**Description** This loop doesn't scale. In case active options would be sorted or grouped by maturity, it would be easy to find an expiring option..
**Client Comment** It would be alternative way to group by maturity. However, the available maturities are infinite, making grouping impossible.

Listing 133:

```
132 for(uint256 i=0;i<activeOptions.length();i++){
```

## 3.134 CVF-134

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

**Description** The expression "activeOptions.at(i)" is calculated twice.
**Recommendation** Consider calculating once and reusing.

Listing 134:

```
133 if(isExpiring(uint256(activeOptions.at(i)))){
      _id = uint256(activeOptions.at(i));
```

## 3.135 CVF-135

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

**Description** These functions should emit some events.
**Client Comment** Added two events 'StampNewOption' and 'StampExpire'.

Listing 135:

```
138 function stampActiveOption(uint256 _id, address _holder)
    ↪ external onlyRole(EXCHANGE_ROLE) {

149 function stampExpiredOption(uint256 _id) external onlyRole(
    ↪ EXCHANGE_ROLE){
```

## 3.136 CVF-136

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

**Description** The expression "optionsList[_id]" is calculated several times.
**Recommendation** Consider calculating once and reusing.

Listing 136:

```
139 optionsList[_id].effectiveTime = block.timestamp;
140 optionsList[_id].maturity = optionsList[_id].effectiveTime +
    ↪ optionsList[_id].tenor;
    optionsList[_id].status = OptionLibrary.OptionStatus.Active;
```

## 3.137 CVF-137

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

**Description** The value "optionsList[_id].effectiveTime" is read from the storage right after it was written into it.
**Recommendation** Consider using "block.timestamp" instead.

Listing 137:

```
139 optionsList[_id].effectiveTime = block.timestamp;
140 optionsList[_id].maturity = optionsList[_id].effectiveTime +
    ↪ optionsList[_id].tenor;
```

## 3.138 CVF-138

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

**Description** The expression "optionsList[_id]" is calculated several times.
**Recommendation** Consider calculating once and reusing.

Listing 138:

```
150 optionsList[_id].exerciseTime = block.timestamp;
    optionsList[_id].status = OptionLibrary.OptionStatus.Expired;
    activeOptionsPerOwner[optionsList[_id].holder].remove(_id);
```

## 3.139 CVF-139

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

**Description** Inside these calls, the underlying decimals number is obtained.
**Recommendation** Consider obtaining the underlying decimals once in the constructor and storing in an internal variable for future use.

Listing 139:

```
159 _tradeUnderlyingAmount = (_aggregateDelta >= 0? _aggregateDelta:
      ↪ int256(0)) - int256(MarketLibrary.balanceDef(underlying,
      ↪ _address));

161 _tradeUnderlyingAmount = MarketLibrary.cvtDecimalsInt(
      ↪ _tradeUnderlyingAmount, underlying);}

179   _repayAmount = MarketLibrary.cvtDecimals(_repayAmount,
        ↪ underlying);}}
```

## 3.140 CVF-140

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

**Description** Inside these calls, the funding decimals number is obtained.
**Recommendation** Consider obtaining the funding decimals once in the constructor and storing in an internal variable for future use.

Listing 140:

```
160 _tradeFundingAmount = MarketLibrary.cvtDecimalsInt(OptionLibrary
    ↪ .getOpposeTrade(_tradeUnderlyingAmount, _price,
    ↪ _swapSlippage), funding);

178 _repaySwapValue = MarketLibrary.cvtDecimals(MulDiv(
    ↪ _repayAmount, _price, OptionLibrary.Multiplier()),
    ↪ funding);
```

## 3.141 CVF-141

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** OptionVault.sol

**Description** There should be named constants or even enum for the valid lending pool rate modes.
**Client Comment** Leave it as it is.

Listing 141:

```
173 (int256 _loanTradeAmount, , ) = MarketLibrary.getLoanTrade(
    ↪ _address, ILendingPoolAddressesProvider(aaveAddress).
    ↪ getAddress("0x1"), _aggregateDelta, underlying,
    ↪ _lendingPoolRateMode == 2);

187 (_loanTradeAmount, _targetLoan, _loanAddress) = MarketLibrary.
    ↪ getLoanTrade(_address, _protocolAds, _aggregateDelta,
    ↪ underlying, _lendingPoolRateMode == 2);
```

## 3.142 CVF-142

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

**Description** The expression "uint256(-_loanTradeAmount)" is calculated twice.
**Recommendation** Consider calculating once and reusing.

Listing 142:

```
177 _repayAmount = uint256(-_loanTradeAmount) - Math.min(uint256(-
    ↪ _loanTradeAmount), ERC20(underlying).balanceOf(address(
    ↪ this)));
```

## 3.143 CVF-143

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

**Description** Should be "^0.8.0".

Listing 143:

```
2 pragma solidity 0.8.10;
```

## 3.144 CVF-144

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

**Description** These variables should be declared as internal.

Listing 144:

```
15 AggregatorV3Interface internal priceInterface;
   bytes32 public tokenHash;
   string public description;

23 uint256 internal priceMultiplier;
   uint256 internal quoteAdjustment;

29 uint256 public volParamDecimals;
30 uint256 internal parameterMultiplier;
```

## 3.145 CVF-145

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

**Description** This variable is not used.
**Recommendation** Consider removing it.

Listing 145:

```
22 uint256 internal priceDecimals;
```

## 3.146 CVF-146

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

**Description** The type of the "_priceSourceId" argument should be "AggregatorV3Interface".

Listing 146:

```
32 constructor( address _priceSourceId, uint256 _parameterDecimals,
   ↪ string memory _tokenName ) {
```

## 3.147 CVF-147

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

**Description** The "abi.encodePacked" call is redundant here. Just do: keccak256(bytes(_tokenName))

Listing 147:

```
35 tokenHash = keccak256(abi.encodePacked(_tokenName));
```

## 3.148 CVF-148

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

**Description** The "priceInterface.decimals()" expression is calculated twice.
**Recommendation** Consider calculating once and reusing.

Listing 148:

```
42 priceMultiplier = 10 ** priceInterface.decimals();
   quoteAdjustment = 10 ** (18 - priceInterface.decimals());
```

## 3.149 CVF-149

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** VolatilityChain.sol

**Description** This will revert in case priceInterface.decimals() > 18.
**Recommendation** Consider refactoring the code to support such a case as well.
**Client Comment** Removed and replaced with 'FullMath.ethdiv'.

Listing 149:

```
43 quoteAdjustment = 10 ** (18 - priceInterface.decimals());
```

## 3.150 CVF-150

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

**Description** This could be calculated as: 1e18 / priceMultiplier
**Client Comment** Removed and replaced with 'FullMath.ethdiv'.

Listing 150:

```
43 quoteAdjustment = 10 ** (18 - priceInterface.decimals());
```

## 3.151 CVF-151

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** VolatilityChain.sol

**Description** This function reverts in case there are no tenors. Probably not an issue.
**Client Comment** The missing tenors would be interpolated within the function.

Listing 151:

```
47 function getVol(uint256 _tenor) external override view returns(
   ↪ uint256 _vol){
```

## 3.152 CVF-152

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

**Description** Should be "else {".

Listing 152:

```
50 if(!tenors.contains(_tenor)){
```

## 3.153 CVF-153

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

**Description** The "_upperVol" and "_lowerVol" variables may be updated several times.
**Recommendation** Consider updating once after the loop.

Listing 153:

```
57 if(_tenorI>_tenor && (_tenorI < _upperTenor || _upperTenor ==
   ↪ 0) ){ _upperTenor = _tenorI; _upperVol = priceBook[_tenorI
   ↪ ][latestBookTime[_tenorI]].volatility; }
   if(_tenorI<_tenor && (_tenorI > _lowerTenor || _lowerTenor ==
   ↪ 0) ){ _lowerTenor = _tenorI; _lowerVol = priceBook[_tenorI
   ↪ ][latestBookTime[_tenorI]].volatility; }}
```

## 3.154 CVF-154

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

**Description** Should be "else if (...)".

Listing 154:

```
58    if(_tenorI <_tenor && (_tenorI > _lowerTenor || _lowerTenor ==
        ↪  0) ){ _lowerTenor = _tenorI; _lowerVol = priceBook[
        ↪ _tenorI][latestBookTime[_tenorI]].volatility; }}

60  if(_lowerTenor==0) {_vol = _upperVol;}
    if(_upperTenor >0 && _lowerTenor >0){ _vol = MulDiv(_lowerVol,
       ↪ _upperTenor - _tenor, _upperTenor - _lowerTenor) + MulDiv(
       ↪ _upperVol, _tenor - _lowerTenor, _upperTenor - _lowerTenor
       ↪ );}}
```

## 3.155 CVF-155

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

**Description** Should be "else if (...)".

Listing 155:

```
60  if(_lowerTenor==0) {_vol = _upperVol;}
```

## 3.156 CVF-156

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

**Description** This expression "_upperTenor - _lowerTenor" is calculated twice.
**Recommendation** Consider calculating once and reusing.

Listing 156:

```
61  if(_upperTenor >0 && _lowerTenor >0){ _vol = MulDiv(_lowerVol,
       ↪ _upperTenor - _tenor, _upperTenor - _lowerTenor) + MulDiv(
       ↪ _upperVol, _tenor - _lowerTenor, _upperTenor - _lowerTenor
       ↪ );}}
```

## 3.157 CVF-157

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

**Description** Should be "else".

Listing 157:

```
61 if(_upperTenor >0 && _lowerTenor >0){ _vol = MulDiv(_lowerVol,
   ↪ _upperTenor - _tenor, _upperTenor - _lowerTenor) + MulDiv(
   ↪ _upperVol, _tenor - _lowerTenor, _upperTenor - _lowerTenor
   ↪ );}}
```

## 3.158 CVF-158

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

**Description** Underflow is possible when converting "_price" to "uint256".
**Recommendation** Consider using safe conversion.

Listing 158:

```
66 return (uint256(_price) * quoteAdjustment, uint256(_timeStamp))
   ↪ ;}

70 uint256 _updatePrice = uint256(_price);

103 uint256 _updatePrice = uint256(_price);
```

## 3.159 CVF-159

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

**Description** The conversion of "_timeStamp" to "uint256" is redundant, as this variable is already has type "uint".

Listing 159:

```
66 return (uint256(_price) * quoteAdjustment, uint256(_timeStamp))
   ↪ ;}
```

## 3.160 CVF-160

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

**Description** Underflow is possible when converting "_price" to "uint256".
**Recommendation** Consider using safe conversion.

Listing 160:

```
70 uint256 _updatePrice = uint256(_price);
```

## 3.161 CVF-161

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

**Description** The expression "tenor.length" is calculated on every loop iteration.
**Recommendation** Consider calculating once and reusing.

Listing 161:

```
72 for(uint i = 0;i< tenors.length();i++) {
```

## 3.162 CVF-162

- **Severity** Moderate
- **Status** Info
- **Category** Suboptimal
- **Source** VolatilityChain.sol

**Description** This loop doesn't scale. In case of too many tenors it could not fit into the block gas limit.
**Recommendation** Consider implementing an ability to update only some of the tenors.
**Client Comment** The number of tenors are controlled by the deployer address only. The number is intended to be only limited (below 5). Also the update function is only executed and monitored from limited amount of addresses, which allows them to adjust gas in case the number of tenors increases.

Listing 162:

```
72 for(uint i = 0;i< tenors.length();i++) {
```

## 3.163 CVF-163

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

**Description** The expression "latestBookTime[_tenor]" is calculated several times.
**Recommendation** Consider calculating once and reusing.

Listing 163:

```
74  if(_updatePrice > priceBook[_tenor][latestBookTime[_tenor]].
        ↪ highest){
      priceBook[_tenor][latestBookTime[_tenor]].highest=_updatePrice
        ↪ ;}
    if(_updatePrice<priceBook[_tenor][latestBookTime[_tenor]].lowest
        ↪ ){
       priceBook[_tenor][latestBookTime[_tenor]].lowest=_updatePrice
        ↪ ;}

79  if(_timeStamp>= (latestBookTime[_tenor] + _tenor)){
80    priceBook[_tenor][latestBookTime[_tenor]].endTime = _timeStamp
        ↪ ;
      priceBook[_tenor][latestBookTime[_tenor]].close = _updatePrice
        ↪ ;
      uint256 _periodMove = (priceBook[_tenor][latestBookTime[_tenor
        ↪ ]].open < _updatePrice)? (MulDiv(_updatePrice,
        ↪ priceMultiplier, priceBook[_tenor][latestBookTime[_tenor
        ↪ ]].open) - priceMultiplier) : (priceMultiplier - MulDiv(
        ↪ _updatePrice, priceMultiplier, priceBook[_tenor][
        ↪ latestBookTime[_tenor]].open ));
      uint256 _largestMove = Math.max(MulDiv(priceBook[_tenor][
        ↪ latestBookTime[_tenor]].highest, priceMultiplier,
        ↪ priceBook[_tenor][latestBookTime[_tenor]].open) -
        ↪ priceMultiplier , priceMultiplier - MulDiv(priceBook[
        ↪ _tenor][latestBookTime[_tenor]].lowest, priceMultiplier,
        ↪  priceBook[_tenor][latestBookTime[_tenor]].open ));
```

(... 86, 90)

## 3.164 CVF-164

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

**Description** The expression "priceBook[_tenor][latestBookTime[_tenor]]" is calculated several times.
**Recommendation** Consider calculating once and reusing.

Listing 164:

```
74  if(_updatePrice > priceBook[_tenor][latestBookTime[_tenor]].
      ↪ highest){
    priceBook[_tenor][latestBookTime[_tenor]].highest=_updatePrice
      ↪ ;}
    if(_updatePrice<priceBook[_tenor][latestBookTime[_tenor]].lowest
      ↪ ){
     priceBook[_tenor][latestBookTime[_tenor]].lowest=_updatePrice
      ↪ ;}

80    priceBook[_tenor][latestBookTime[_tenor]].endTime = _timeStamp
      ↪ ;
    priceBook[_tenor][latestBookTime[_tenor]].close = _updatePrice
      ↪ ;
    uint256 _periodMove = (priceBook[_tenor][latestBookTime[_tenor
      ↪ ]].open < _updatePrice)? (MulDiv(_updatePrice,
      ↪ priceMultiplier, priceBook[_tenor][latestBookTime[_tenor
      ↪ ]].open) - priceMultiplier) : (priceMultiplier - MulDiv(
      ↪ _updatePrice, priceMultiplier, priceBook[_tenor][
      ↪ latestBookTime[_tenor]].open ));
    uint256 _largestMove = Math.max(MulDiv(priceBook[_tenor][
      ↪ latestBookTime[_tenor]].highest, priceMultiplier,
      ↪ priceBook[_tenor][latestBookTime[_tenor]].open) -
      ↪ priceMultiplier , priceMultiplier - MulDiv(priceBook[
      ↪ _tenor][latestBookTime[_tenor]].lowest, priceMultiplier,
      ↪  priceBook[_tenor][latestBookTime[_tenor]].open ));
```

(... 86)

## 3.165 CVF-165

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

**Description** The expression "priceBook[_tenor][latestBookTime[_tenor]].highest" is calculated several times.
**Recommendation** Consider calculating once and reusing.
**Client Comment** Left it as it is as impact is small.

Listing 165:

```
74 if(_updatePrice > priceBook[_tenor][latestBookTime[_tenor]].
     ↪ highest){
   priceBook[_tenor][latestBookTime[_tenor]].highest=_updatePrice
     ↪ ;}

83   uint256 _largestMove = Math.max(MulDiv(priceBook[_tenor][
     ↪ latestBookTime[_tenor]].highest, priceMultiplier,
     ↪ priceBook[_tenor][latestBookTime[_tenor]].open) -
     ↪ priceMultiplier, priceMultiplier - MulDiv(priceBook[
     ↪ _tenor][latestBookTime[_tenor]].lowest, priceMultiplier,
     ↪  priceBook[_tenor][latestBookTime[_tenor]].open ));
```

## 3.166 CVF-166

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

**Description** The expression "priceBook[_tenor][latestBookTime[_tenor]].lowest" is calculated several times.
**Recommendation** Consider calculating once and reusing.
**Client Comment** Left it as it is as impact is small.

Listing 166:

```
76 if(_updatePrice<priceBook[_tenor][latestBookTime[_tenor]].lowest
     ↪ ){
    priceBook[_tenor][latestBookTime[_tenor]].lowest=_updatePrice
     ↪ ;}

83   uint256 _largestMove = Math.max(MulDiv(priceBook[_tenor][
     ↪ latestBookTime[_tenor]].highest, priceMultiplier,
     ↪ priceBook[_tenor][latestBookTime[_tenor]].open) -
     ↪ priceMultiplier, priceMultiplier - MulDiv(priceBook[
     ↪ _tenor][latestBookTime[_tenor]].lowest, priceMultiplier,
     ↪  priceBook[_tenor][latestBookTime[_tenor]].open ));
```

## 3.167 CVF-167

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

**Description** The expression "priceBook[_tenor][latestBookTime[_tenor]].open" is calculated several times.
**Recommendation** Consider calculating once and reusing.

Listing 167:

```
82 uint256 _periodMove = (priceBook[_tenor][latestBookTime[_tenor
    ↪ ]].open < _updatePrice)? (MulDiv(_updatePrice,
    ↪ priceMultiplier, priceBook[_tenor][latestBookTime[_tenor
    ↪ ]].open) - priceMultiplier) : (priceMultiplier - MulDiv(
    ↪ _updatePrice, priceMultiplier, priceBook[_tenor][
    ↪ latestBookTime[_tenor]].open ));
   uint256 _largestMove = Math.max(MulDiv(priceBook[_tenor][
    ↪ latestBookTime[_tenor]].highest, priceMultiplier,
    ↪ priceBook[_tenor][latestBookTime[_tenor]].open) -
    ↪ priceMultiplier , priceMultiplier - MulDiv(priceBook[
    ↪ _tenor][latestBookTime[_tenor]].lowest, priceMultiplier,
    ↪ priceBook[_tenor][latestBookTime[_tenor]].open ));
```

## 3.168 CVF-168

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

**Description** The expression "volatilityParameters[_tenor]" is calculated several times.
**Recommendation** Consider calculating once and reusing.

Listing 168:

```
86 priceBook[_tenor][_timeStamp].volatility = Sqrt(
   ↪ volatilityParameters[_tenor].ltVolWeighted + MulDiv(
   ↪ _periodMove * _periodMove, volatilityParameters[_tenor].q,
   ↪  parameterMultiplier) + MulDiv(priceBook[_tenor][
   ↪ latestBookTime[_tenor]].volatility * priceBook[_tenor][
   ↪ latestBookTime[_tenor]].volatility, volatilityParameters[
   ↪ _tenor].p, parameterMultiplier));
   priceBook[_tenor][_timeStamp].accentus = Sqrt(
   ↪ volatilityParameters[_tenor].ltVolWeighted + MulDiv(
   ↪ _largestMove * _largestMove, volatilityParameters[_tenor].
   ↪ q, parameterMultiplier) + MulDiv(priceBook[_tenor][
   ↪ latestBookTime[_tenor]].accentus * priceBook[_tenor][
   ↪ latestBookTime[_tenor]].accentus, volatilityParameters[
   ↪ _tenor].p, parameterMultiplier));
```

## 3.169 CVF-169

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

**Description** The expression "volatilityParameters[_tenor]" is calculated several times.
**Recommendation** Consider calculating once and reusing.
**Client Comment** Left it as it is, as impact is small.

Listing 169:

```
99  volatilityParameters[_tenor] = _volParams;
100 volatilityParameters[_tenor].ltVolWeighted = MulDiv(
    ↪ volatilityParameters[_tenor].ltVol*volatilityParameters[
    ↪ _tenor].ltVol, volatilityParameters[_tenor].w,
    ↪ parameterMultiplier);

107 priceBook[_tenor][_timeStamp].volatility = priceBook[_tenor][
    ↪ _timeStamp].accentus = volatilityParameters[_tenor].
    ↪ initialVol;
```

## 3.170 CVF-170

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

**Description** The expression "priceBook[_tenor][_timeStamp]" is calculated several times.
**Recommendation** Consider calculating once and reusing.

Listing 170:

```
106 priceBook[_tenor][_timeStamp].startTime = _timeStamp;
    priceBook[_tenor][_timeStamp].volatility = priceBook[_tenor][
       ↪ _timeStamp].accentus = volatilityParameters[_tenor].
       ↪ initialVol;
    priceBook[_tenor][_timeStamp].open = priceBook[_tenor][
       ↪ _timeStamp].highest = priceBook[_tenor][_timeStamp].lowest
       ↪  = _updatePrice;}
```

## 3.171 CVF-171

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

**Description** This function should emit some event.

Listing 171:

```
110 function removeTenor(uint256 _tenor) external onlyOwner{
```

## 3.172 CVF-172

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** VolatilityChain.sol

**Description** For a non existing tenor this function silently returns an empty price book.
**Recommendation** Consider reverting in such a case
**Client Comment** Added require clause.

Listing 172:

```
114 function getPriceBook(uint256 _tenor) external view returns(
       ↪ PriceStamp memory){
```

## 3.173 CVF-173

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

**Description** These functions are redundant as the corresponding variables are already public. Just rename the functions in the interface to match the variables names.

Listing 173:

```
118 function getTokenHash() external override view returns(bytes32){
    ↪ return tokenHash;}
function getDecription() external override view returns (string
    ↪ memory) {return description;}}
```

## 3.174 CVF-174

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

**Description** Should be "^0.8.0".

Listing 174:

```
2 pragma solidity 0.8.10;
```

## 3.175 CVF-175

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

**Description** This role is not used.
**Recommendation** Consider removing it.

Listing 175:

```
9 bytes32 public constant ADMIN_ROLE = keccak256("ADMIN_ROLE");
```

## 3.176 CVF-176

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

**Description** This constant is not used.
**Recommendation** Consider removing it.

Listing 176:

```
11 uint256 private constant ethMultiplier = 10 ** 18;
```

## 3.177 CVF-177

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

**Description** The "abi.encodePacked" call is redundant. Just do: keccak256(bytes(_tokenName))

Listing 177:

```
19 tokenHash = keccak256(abi.encodePacked(_tokenName));}
```

## 3.178 CVF-178

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** VolatilityToken.sol

**Description** This function allows burning tokens from arbitrary accounts. This looks dangerous.
**Recommendation** Consider allowing burning only caller's token or token explicitly approved to the caller.
**Client Comment** Allowance check is added.

Listing 178:

```
22 function burn(address _account, uint256 _amount) public onlyRole
   ↪ (EXCHANGE_ROLE){ _burn(_account, _amount);}
```