ABDK CONSULTING

SMART CONTRACT AUDIT

Risk Harbor

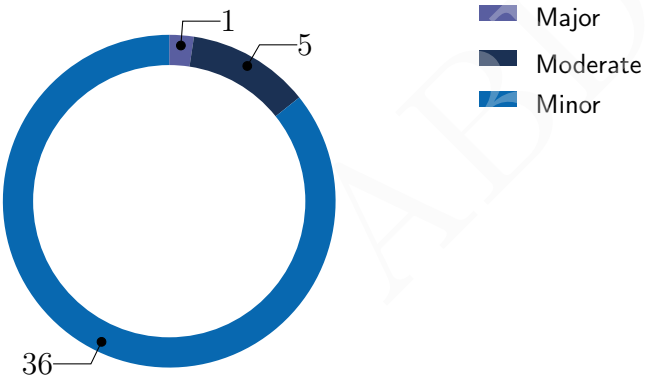
Solidity v.1.2.1

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
19rd June 2021

We've been asked to review the Risk Harbor smart contracts given in a private repository. We have identified many issues, but all severe ones have been fixed.



Findings

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

ID	Severity	Category	Status
CVF-28	Minor	Suboptimal	Fixed
CVF-29	Minor	Suboptimal	Fixed
CVF-30	Minor	Suboptimal	Fixed
CVF-31	Minor	Suboptimal	Fixed
CVF-32	Minor	Suboptimal	Fixed
CVF-33	Minor	Suboptimal	Opened
CVF-34	Minor	Suboptimal	Fixed
CVF-35	Minor	Bad naming	Fixed
CVF-36	Minor	Readability	Fixed
CVF-37	Major	Flaw	Fixed
CVF-38	Minor	Flaw	Fixed
CVF-39	Minor	Suboptimal	Fixed
CVF-40	Minor	Bad naming	Fixed
CVF-41	Minor	Procedural	Fixed
CVF-42	Minor	Bad naming	Fixed

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	June 18, 2021	D. Khovratovich	Initial Draft
0.2	June 18, 2021	D. Khovratovich	Minor revision
1.0	June 19, 2021	D. Khovratovich	Release
1.1	June 23, 2021	D. Khovratovich	Add client comment, status update
2.0	June 19, 2021	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 audited the Risk Harbor Github repository with tag 1.2.1. Concretely, the following files were audited:

- Consumer/Consumer.sol;
- Consumer/Dripper.sol;
- Defaultector/Abstracts/IBase.sol;
- Defaultector/Abstracts/ILendingMarket.sol;
- Defaultector/Implementations/AaveWrapper.sol;
- Defaultector/Implementations/CompoundWrapper.sol;
- Defaultector/Implementations/YearnWrapper.sol;
- Interfaces/IConsumer.sol;
- Interfaces/IUnderwriter.sol;
- Shared/Shared.sol;
- Storage/Storage.sol;
- Underwriter/Claims.sol;
- Underwriter/Conversion.sol;
- Underwriter/MarketStatus.sol;
- Underwriter/Shares.sol;
- Underwriter/Underwriter.sol;
- Underwriter/Utilization.sol.

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** Underwriter.sol MarketStatus.sol Utilization.sol Shared.sol Shares.sol Conversion.sol Claims.sol Storage.sol IAave.sol AaveWrapper.sol CompoundWrapper.sol Dripper.sol ILendingMarket.sol IBase.sol Consumer.sol ErrorReporter.sol IUnderwriter.sol IPolicyHolder.sol IDefaultector.sol IYearn.sol

Recommendation Should be "ĝ0.8.0" rather than "0.8.0".

Listing 1:

```
4 solidity 0.8.0;
```

3.2 CVF-2

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

Recommendation These checks could be simplified as: IERC20(0) != _creditToken IERC20(0) != _underlyingToken IDefaultector(0) != _defaultector IERC20(0) != _currencyToken

Listing 2:

```
116 address(0) != address(_creditToken),

120 address(0) != address(_underlyingToken),

124 address(0) != address(_defaultector),

128 address(0) != address(_currencyToken),
```

3.3 CVF-3

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

Recommendation It is bad practice to apply state-modifying modifiers before the reentrancy protection.

Listing 3:

```
170 _atStatus(Status.OPEN)
    nonReentrant
```

3.4 CVF-4

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

Description The address of "atPricePoint[_pricePoint] is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 4:

```
216 (atPricePoint[_pricePoint].liquidity -
        atPricePoint[_pricePoint].utilizedLiquidity),
```

3.5 CVF-5

- **Severity** Minor
- **Status** Opened
- **Category** Flaw
- **Source** Underwriter.sol

Recommendation There is no explicit range check for the "_pricePoint" value. Also, in other places, price point has to be > 0 and < 100, i.e. the valid range is actually 1..99.
Client Comment Will update when we go through and update the code for public release.

Listing 5:

```
269 * @param _pricePoint Price Point [0-100]
```

3.6 CVF-6

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

Description There is no actual check here.
Recommendation Consider adding an explicit check that amount is greater than zero.

Listing 6:

```
289 // Check if they called startSwap before
290 uint256 amount = startedClaims[msg.sender].amount;
```

3.7 CVF-7

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** Underwriter.sol

Description The returned value is ignored.
Recommendation Consider using the "safeTransfer" function.

Listing 7:

```
310 creditToken.transfer(address(defaultector), amount);
```

3.8 CVF-8

- **Severity** Moderate
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Underwriter.sol

Description So in the closed state, a pending swap can neither be finished nor cancelled, so the tokens taken from the user then the swap was started are effectively lost.

Listing 8:

```
334 function cancelSwap() external nonReentrant _notAtStatus(Status.
    ↪ CLOSED) {
```

3.9 CVF-9

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

Recommendation This code runs even if the amount is 0.

Listing 9:

```
337 creditToken.safeTransfer(msg.sender, withdrawable);
    emit ClaimCancel(msg.sender, withdrawable);
```

3.10 CVF-10

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

Recommendation This could be simplified to: if (atStatus(Status.OPEN)) {

Listing 10:

```
352 if (!atStatus(Status.OPEN)) {} else {
```

3.11 CVF-11

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

Recommendation This could be simplified to: if (underwriterPoolCurrencyTokenLimit != _amount) { underwriterPoolCurrencyTokenLimit = _amount; ...
Client Comment Not in code anymore.

Listing 11:

```
365 uint256 oldLimit = underwriterPoolCurrencyTokenLimit;
    underwriterPoolCurrencyTokenLimit = _amount;
    if (_amount != oldLimit) {
```

3.12 CVF-12

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

Recommendation This could be simplified as: require (IDefaultector (0) != _newDefaultector);

Listing 12:

```
392 require(address(0) != address(_newDefaulector));
    IDefaultector oldDefaultector = defaultector;
     defaultector = _newDefaulector;

396 if (defaultector != oldDefaultector) {
```

3.13 CVF-13

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

Recommendation This function should probably log some event.
Client Comment Not in code anymore.

Listing 13:

```
408 function emergencyDump() external onlyOwner {
```

3.14 CVF-14

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

Description Returned values are returned here. Probably not an issue.

Listing 14:

```
411 currencyToken.transfer(

417 creditToken.transfer(
```

3.15 CVF-15

- **Severity** Moderate
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Underwriter.sol

Description This function allows the owner to steal all the tokens form the smart contract at any time.
Recommendation Consider putting some restrictions, otherwise the protocol doesn't seem trustless.
Client Comment Not in code anymore.

Listing 15:

```
411 currencyToken.transfer(

417 creditToken.transfer(
```

3.16 CVF-16

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

Recommendation This could be simplified as: if (value != newValue) { value = newValue; emit (...); }

Listing 16:

```
425 uint256 oldPayoutRatio = payoutRatio;
    payoutRatio = _payoutRatio;
    if (oldPayoutRatio != payoutRatio) {

433 uint256 oldDefaultRatio = defaultRatio;
    defaultRatio = _defaultRatio;
    if (oldDefaultRatio != defaultRatio) {
```

3.17 CVF-17

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

Recommendation This function should emit some event.

Listing 17:

```
443 function updatePolicyholder(IPolicyHolder _policyholder)
```

3.18 CVF-18

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

Recommendation This could be simplified as: if (marketStatus != _status) { marketStatus = _status; ...

Listing 18:

```
43  Status oldStatus = marketStatus;
    marketStatus = _status;

46  if (oldStatus != marketStatus) {
```

3.19 CVF-19

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

Recommendation Logging the first variable is probably redundant as it has been already logged as a new status.

Listing 19:

```
47  emit StatusChange(oldStatus, _status);
```

3.20 CVF-20

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

Description The address of "atPricePoint[_pricePoint]" is calculated several times.
Recommendation Consider calculating once and reusing.

Listing 20:

```
85  atPricePoint[_pricePoint].liquidity -
    atPricePoint[_pricePoint].utilizedLiquidity;

116 atPricePoint[_pricePoint].utilizedLiquidity >=
    atPricePoint[_pricePoint].liquidity;
```

3.21 CVF-21

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

Description The address of "atPricePoint[_pricePoint]" is calculate several times.
Recommendation Consider calculating once and reusing.

Listing 21:

```
65  uint256 balance = atPricePoint[_pricePoint].liquidity;

73      creditToCurrency(atPricePoint[_pricePoint].drippedPremiums);

95  uint256 _shares = atPricePoint[_pricePoint].shares;

108 atPricePoint[_pricePoint].shares += _shares;

111 atPricePoint[_pricePoint].liquidity += _amount;

148     atPricePoint[_pricePoint].liquidity.mulDiv(_shares,
        ↪ numShares);

159     atPricePoint[_pricePoint].drippedPremiums.mulDiv(

171 atPricePoint[_pricePoint].liquidity -= withdrawableCapital;
    atPricePoint[_pricePoint].drippedPremiums -=
      ↪ withdrawablePremiums;
    atPricePoint[_pricePoint].shares -= _shares;
```

3.22 CVF-22

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

Description Phantom overflow is possible here, i.e. a situation when the final result would fit into the destination type, but some intermediary calculations overflow.
Recommendation Consider using "muldiv" function.

Listing 22:

```
103 _shares = (_amount * _shares) / calculateValue(_pricePoint);
```

3.23 CVF-23

- **Severity** Moderate
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Shares.sol

Description The "approve" function overwrites the current allowance rather than adds to it. Thus, the second "approve" call will effectively cancel any not yet used allowance.
Recommendation Consider adding to the current allowance like this: 'token.approve (msg.sender, token.allowance (address(this), msg.sender) + extraAllowance);'

Listing 23:

```
179 // Use approval as a temporary storage if transfer fails
180 // User can transfer themselves then
    currencyToken.approve(msg.sender, withdrawableCapital);

192 // Use approval as a temporary storage if transfer fails
    // User can transfer themselves then
    creditToken.approve(msg.sender, withdrawablePremiums);
```

3.24 CVF-24

- **Severity** Moderate
- **Status** Fixed
- **Category** Flaw
- **Source** Shares.sol

Description The returned value is ignored here, thus a failed transfer could be counter as a successful one.
Recommendation Consider using "safeTransfer" function.
Client Comment Created an external function to wrap 'safeTransfer', which is an internal method. If it fails, give the user allowance.

Listing 24:

```
211 _token.transfer(_to, _amount);
```

3.25 CVF-25

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

Description This variable is redundant.
Recommendation Consider turning it into a compile-time constant or just using a hardcoded value.

Listing 25:

```
88 uint256 decimals = 1e18;
```

3.26 CVF-26

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

Description The address of "atPricePoint[_pricePoint]" is calculated three times.
Recommendation Consider calculating once and reusing like this: PricePointParams storage pricePointParams = atPricePoint [_pricePoint];

Listing 26:

```
53 atPricePoint[_pricePoint].latestPaymentBlock = block.number;

57 atPricePoint[_pricePoint].undrippedPremiums -= premiumDripAmt;

72     atPricePoint[_pricePoint].latestPaymentBlock
```

3.27 CVF-27

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

Recommendation Logging this variable is redundant as it has been already logged as 'latestPaymentBlock'
Client Comment I removed the unused variable.

Listing 27:

```
70 oldPaymentBlock,
```

3.28 CVF-28

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

Description 'remainingTime' can't be 0 at this point due to the if condition above

Listing 28:

```
109 if (remainingTime == 0 || bal == 0) return 0;
```

3.29 CVF-29

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

Recommendation This variable is redundant. Turn it into a compile-time constant, or just use a hardcoded value.
Client Comment Also did this for 'PoolTogether' wrapper.

Listing 29:

```
59 uint256 decimals = 1e18;
```

3.30 CVF-30

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

Recommendation This could be simplified as: return (pulledOut * decimals < baselineAmount, pulledOut);
Client Comment Also did this for 'PoolTogether' wrapper.

Listing 30:

```
62 if (pulledOut * decimals < baselineAmount) return (true,
     ↪ pulledOut);
   return (false, pulledOut);
```

3.31 CVF-31

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

Recommendation These variables are redundant. Turn it into a compile-time constant or just use a hardcoded value instead.

Listing 31:

```
46 uint256 decimals = 1e18;

91 uint256 decimals = 1e18;
```

3.32 CVF-32

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

Recommendation This could be simplified as: return (pulledOut * decimals < baselineAmount < pulledOut);

Listing 32:

```
92 if (pulledOut * decimals < baselineAmount) {
       return (true, pulledOut);
   } else {
       return (false, pulledOut);
   }
```

3.33 CVF-33

- **Severity** Minor
- **Status** Opened
- **Category** Suboptimal
- **Source** AaveWrapper.sol

Recommendation These two constants have the same value. One of them is probably redundant.
Client Comment Can't get rid of one of them since one is used in testing and the other one is what we actually use

Listing 33:

```
18 address public constant mainnetLendingPool =

25 ILendingPoolAddressesProvider public constant provider =
```

3.34 CVF-34

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

Recommendation This could be simplified as: return (pulledOut * 1e18 < baselineAmount, pulledOut);

Listing 34:

```
99  if (pulledOut * 1e18 < baselineAmount) {
100     return (true, pulledOut);
    } else {
        return (false, pulledOut);
    }
```

3.35 CVF-35

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

Recommendation Once the contract was renamed, the file should be renamed as well.

Listing 35:

```
37 PolicyHolder is
```

3.36 CVF-36

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

Recommendation This could be written as: IERC20(0) != _creditToken IERC20(0) != _currencyToken IUnderwriter(0) != _underwriter

Listing 36:

```
77 address(0) != address(_creditToken),

81 address(0) != address(_currencyToken),

85 address(0) != address(_underwriter),
```

3.37 CVF-37

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

Description This is not a correct way to do 'muldiv' rounding up. The correct way to calcualte x * y / z rounding up would be: muldiv (x, y, z) + (mulmod (x, y, z) > 0 ? 1 : 0)
Client Comment Done in Drew's PR.

Listing 37:

```
140 if (PRBMathCommon.mulDiv(_amount, _pricePoint, 10) % 10 != 0) {
        premium = PRBMathCommon.mulDiv(_amount, _pricePoint, 100) +
            ↪ 1;
    } else {
        premium = PRBMathCommon.mulDiv(_amount, _pricePoint, 100);
    }
```

3.38 CVF-38

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

Description There is no check to ensure that the array lengths are the same. Also, a single array of structures with two fields would be more efficient that two parallel arrays and would make the length check unnecessary.

Listing 38:

```
180 uint256[] memory _pricePoints,
    uint256[] memory _amounts
```

3.39 CVF-39

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

Description This calculates the address of "atPricePoint[_pricePoint]" twice.
Recommendation Consider calculating once and reusing.
Client Comment No longer in code.

Listing 39:

```
235 if (atPricePoint[_pricePoint].latestPaymentBlock == 0) {
        atPricePoint[_pricePoint].latestPaymentBlock = block.number;
```

3.40 CVF-40

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

Description The name is too generic.
Recommendation Consider making it more specific, such as “isValidPricePoint” or, which would be more conventional, “onlyValidPricePoint”.

Listing 40:

```
246 modifier inRange(uint256 _pricePoint) {
```

3.41 CVF-41

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

Description It is a good practice to add new constants to the end of the list, to preserve backward compatibility. Probably not an issue in this case.
Client Comment No changes needed.

Listing 41:

```
17 CLAIM_FAILED, // 4
```

3.42 CVF-42

- **Severity** Minor
- **Status** Fixed
- **Category** Bad naming
- **Source** IDefaultector.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.

Listing 42:

```
13 ) external returns (bool, uint256);
```