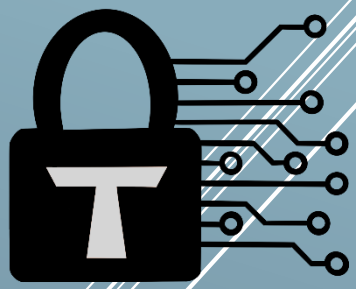
Trust Security

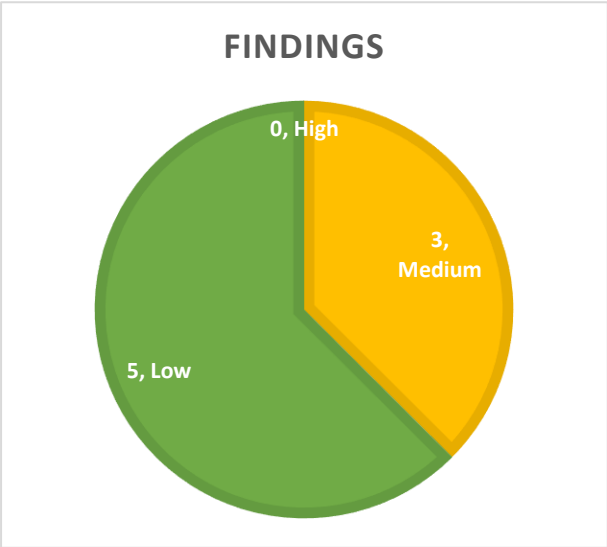


Smart Contract Audit

Rysk Beyond – Inventory upgrade

21/06/23

Executive summary

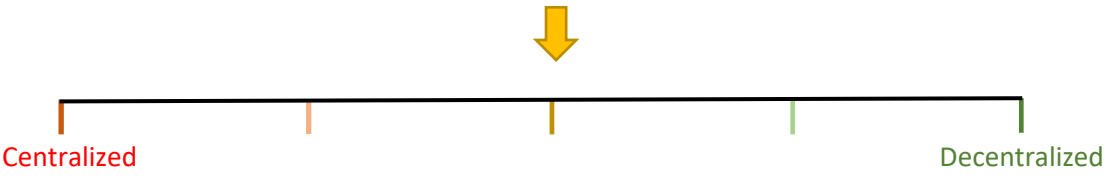


Category	Options
Audited file count	5
Lines of Code	613
Auditor	Trust
Time period	13-21/06/23

Findings

Severity	Total	Fixed	Acknowledged
High	-	-	-
Medium	3	2	1
Low	5	4	1

Centralization score



Signature

Digitally signed by Trust
Date: 2023-06-26 14:17:16

Document properties

Versioning

Version	Date	Description
0.1	21/06/23	Client report
0.2	26/06/23	Mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

Changes in the following files are included in the scope.

- AlphaOptionHandler.sol
- AlphaPortfolioValuesFeed.sol
- BeyondPricer.sol
- OptionExchange.sol
- libraries/OptionsCompute.sol

Repository details

- **Repository URL:** https://github.com/rysk-finance/dynamic-hedging/pull/540
- **Commit hash:** e33c32c80c86e65f06e627a7d5f479661cb5e791
- **Repository URL:** https://github.com/rysk-finance/dynamic-hedging/pull/590
- **Commit hash:** fe5301c6e4e76773a7e9998b4658da118ccb07f2

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Trust is the leading auditor at competitive auditing service Code4rena, reported several critical issues to Immunefi bug bounty platform and is currently a Code4rena judge.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Excellent**	The upgrade has not introduced significant complexity.
Documentation	**Excellent**	The upgrade is documented in detail.
Best practices	**Good**	Project mostly adheres to industry standards.
Centralization risks	**Moderate**	The project multi-sig has far-reaching control.

Findings

Medium severity findings

TRST-M-1 The pricer will revert when pricing options with maximum allowed tenor.

- **Category:** Out-of-bounds errors
- **Source:** BeyondPricer.sol
- **Status:** Fixed

Description

The upgrade introduced tenors, which are a way to customize slippage and spread parameters as a function of time to maturity. The correct tenor is fetched with the function below:

```
function _getTenorIndex(
    uint256 _expiration
) internal view returns (uint16 tenorIndex, int256 remainder) {
    // get the ratio of the square root of seconds to expiry and the
max tenor value in e18 form
    uint unroundedTenorIndex = (((((_expiration - block.timestamp) *
1e18).sqrt()) *
        (numberOfTenors - 1)) / maxTenorValue);
    tenorIndex = uint16(unroundedTenorIndex / 1e18); // always floors
    remainder = int256(unroundedTenorIndex - tenorIndex * 1e18); //
will be between 0 and 1e18
}
```

The **(tenorIndex, remainder)** pair will be sent to the calculation functions, like *_interpolateSlippageGradient()*:

```
function _interpolateSlippageGradient(
    uint16 _tenor,
    int256 _remainder,
    bool _isPut,
    uint256 _deltaBand
) internal view returns (uint80 slippageGradientMultiplier) {
    if (_isPut) {
        int80 y1 =
tenorPricingParams[_tenor].putSlippageGradientMultipliers[_deltaBand]
;
        int80 y2 = tenorPricingParams[_tenor +
1].putSlippageGradientMultipliers[_deltaBand];
        return uint80(int80(y1 + _remainder.mul(y2 - y1)));
    } else {
        int80 y1 =
tenorPricingParams[_tenor].callSlippageGradientMultipliers[_deltaBand
];
        int80 y2 = tenorPricingParams[_tenor +
1].callSlippageGradientMultipliers[_deltaBand];
        return uint80(int80(y1 + _remainder.mul(y2 - y1)));
    }
}
```

The function interpolates two data points, the element in position **_tenor** and in **_tenor + 1**. Note that if **_tenor** is the last index, the second point will overflow the array. This condition will occur when the option tenor is greater than or equal to **maxTenorValue**. The **maxTenorValue** variable is designed to be inclusive, so the function must handle it correctly. Currently, the pricer will refuse to price such options.

Recommended mitigation

If **remainder = 0**, skip the interpolation and return the first data point directly.

Team response

Fixed.

Mitigation review

The three interpolation functions safely handle the case when **remainder = 0**.

TRST-M-2 Black-Scholes formula is not followed for flatIV pricing model

- **Category:** Logical flaws
- **Source:** BeyondPricer.sol
- **Status:** Fixed

Description

The upgrade introduced a special "Flat IV" pricing model for options that have low delta.

```
if (isSell && uint256(delta.abs()) < lowDeltaThreshold) {
    (uint overridePremium, ) = OptionsCompute.quotePriceGreeks(
        _optionSeries,
        isSell,
        bidAskIVSpread,
        riskFreeRate,
        lowDeltaSellOptionFlatIV,
        forward,
        true // override IV
    );
    overridePremium = OptionsCompute.convertToDecimals(
        overridePremium.mul(_amount),
        ERC20(collateralAsset).decimals()
    );
    totalPremium = OptionsCompute.min(totalPremium, overridePremium);
}
```

In this flow, the calculation ignores slippage and spread calculations in favor of vanilla Black-Scholes. However, the calculation also skips an important Black-Scholes parameter called discounting. It is done in the following line when using the standard pricing model:

```
vanillaPremium = vanillaPremium.mul(underlyingPrice).div(forward);
```

Since the **overridePremium** variable is not discounted, the premium will be higher than expected, making the short side profit the difference in expected value while the long loses it.

Recommended mitigation

Add the discount calculation for flatIV model as well.

Team response

Fixed.

Mitigation review

Discounting is now done properly in the flatIV pricing model.

TRST-M-3 Valid orders may not be executable due to TOCTOU of option balance

- **Category:** Race-condition flaws
- **Source:** AlphaOptionHandler.sol
- **Status:** Acknowledged

Description

The inventory code upgrade changed *createOrder()*, so that it will only issue the option series if it cannot facilitate the order execution with the existing balance.

```
if (series == address(0) || ERC20(series).balanceOf(address(this)) <
OptionsCompute.convertToDecimals(_amount, OPYN_DECIMALS)) {
   series = liquidityPool.handlerIssue(_optionSeries);
}
```

Note that at execution-time, if the balance is insufficient the handler will write an option. This assumes it has already been issued.

```
} else {
   if (order.optionSeries.collateral != collateralAsset) {
      revert CustomErrors.CollateralAssetInvalid();
   }
   // write the option contract, includes sending the premium from
the user to the pool, option series should be in e8
   liquidityPool.handlerWriteOption(
      order.optionSeries,
      order.seriesAddress,
      order.amount,
      getOptionRegistry(),
      convertedPrem,
      0, // delta is not used in the liquidityPool unless the oracle
implementation is used, so can be set to 0
      msg.sender
   );
   getPortfolioValuesFeed().updateStores(
      seriesToStore,
      int256(order.amount),
      0,
      order.seriesAddress
   );
}
```

The issue is a TOCTOU (time-of-check, time-of-use) bug regarding the handler's balance. Suppose the contract has 100 oTokens. Two orders are created with an amount of 60 oTokens. These will not trigger an *issue()* because the balance check is passed. At this point, both orders are executed. Only the first one shall be completed successfully. The second order will trigger a *handlerWriteOption()* call which will fail as the series was not issued.

Recommended mitigation

One option is to delay the *issue()* call to execution-time. Another option is to keep track of balance that is already allocated to future executions, and deduct it from the current balance when checking new orders.

Team response

Acknowledged.

Low severity findings

TRST-L-1 Pricing of options far in the future may be incorrect

- **Category:** Overflow issues
- **Source:** BeyondPricer.sol
- **Status:** Fixed

Description

In *_getTenorIndex()*, the index calculation is performed as:

```
uint unroundedTenorIndex = (((((_expiration - block.timestamp) *
1e18).sqrt()) *
    (numberOfTenors - 1)) / maxTenorValue);
tenorIndex = uint16(unroundedTenorIndex / 1e18); // always floors
```

If expiration is a number far in the future, **uint16()** casting may truncate the result and cause a calculation error. As of the current configuration, Rysk handles option approval so it remains unlikely. However, if option minting becomes decentralized, this would open the attack path. The impact would be a significant mispricing of options.

Recommended mitigation

Validate that **unroundedTenorIndex / 1e18 <= 65535**.

Team response

Fixed.

Mitigation review

Fixed using the suggested mitigation.

TRST-L-2 Portfolio will miscalculate exposure when AlphaOptionHandler processes orders with collateral different from the one set

- **Category:** Logical flaws
- **Source:** AlphaOptionHandler.sol
- **Status:** Fixed

Description

The *executeOrder()* function in AlphaOptionHandler allows a user to buy or sell options to or from the pool. The AlphaPortfolioValuesFeed contract holds the current long and short exposure per option series (a tuple that uniquely identifies the underlying, expiry, strike price, collateral and other factors). The upgrade made it so that when the handler has sufficient oTokens, it will not write new options.

```
if (ERC20(order.seriesAddress).balanceOf(address(this)) >=
convertedAmount) {
   // transfer otoken
   SafeTransferLib.safeTransfer(ERC20(order.seriesAddress),
msg.sender, convertedAmount);
   // update stores
   getPortfolioValuesFeed().updateStores(
      seriesToStore,
      0,
      -int256(order.amount),
      order.seriesAddress
   );
   // adjust variables
   liquidityPool.adjustVariables(
      0,
      convertedPrem,
      0,
      true
   );
```

The issue is that the **seriesToStore** structure is built with the Handler's **collateralAsset**, rather than the one fetched from the order.

```
Types.OptionSeries memory seriesToStore = Types.OptionSeries(
   order.optionSeries.expiration,

uint128(OptionsCompute.convertFromDecimals(order.optionSeries.strike,
OPYN_DECIMALS)),
   order.optionSeries.isPut,
   underlyingAsset,
   strikeAsset,
   collateralAsset
);
```

If the **collateralAsset** is different from the optionSeries **collateralAsset**, the wrong asset will be stored in the **storesForAddress** mapping.

```
if (!addressSet.contains(_seriesAddress)) {
   // maybe store them by expiry instead
   addressSet.add(_seriesAddress);
   storesForAddress[_seriesAddress] = OptionStores(_optionSeries,
shortExposure, longExposure);
```

The collateral will never actually be used in the current iteration of the code. Also, a wrong event will be emitted below:

```
emit StoresUpdated(_seriesAddress, shortExposure, longExposure,
_optionSeries);
```

Recommended mitigation

Pass the correct **collateralAsset** when constructing the **seriesToStore** variable.

Team response

Fixed

Mitigation review

Fix has been applied correctly.

TRST-L-3 The "flat IV" parameters are not validated to be safe.

- **Category:** Input validation issues
- **Source:** BeyondPricer.sol
- **Status:** Acknowledged

Description

The following functions are introduced for "flat IV" pricing model.

```
function setLowDeltaSellOptionFlatIV(uint256
_lowDeltaSellOptionFlatIV) external {
    _onlyManager();
    emit LowDeltaSellOptionFlatIVChanged(_lowDeltaSellOptionFlatIV,
lowDeltaSellOptionFlatIV);
    lowDeltaSellOptionFlatIV = _lowDeltaSellOptionFlatIV;
}
function setLowDeltaThreshold(uint256 _lowDeltaThreshold) external {
    _onlyManager();
    emit LowDeltaThresholdChanged(_lowDeltaThreshold,
lowDeltaThreshold);
    lowDeltaThreshold = _lowDeltaThreshold;
}
```

The functions do not restrict the input to sensible values. They should have a safe upper bound to avoid being abused or accidentally set.

Recommended mitigation

Choose a safe upper bound for the set IV and low delta thresholds.

Team response

Acknowledged

TRST-L-4 DHV exposure updates do not emit adequate events

- **Category:** Event emission issues
- **Source:** AlphaPortfolioValuesFeed.sol
- **Status:** Fixed

Description

The upgrade adds an option for governance to update the critical **netDhvExposure** mapping of delta exposures.

```
function setNetDhvExposures(
   bytes32[] memory _optionHashes,
   int256[] memory _netDhvExposures
) external {
   _onlyGovernor();
   uint256 arrayLength = _optionHashes.length;
   require(arrayLength == _netDhvExposures.length);
   for (uint i; i < arrayLength; i++) {
      if (uint256(_netDhvExposures[i].abs()) > maxNetDhvExposure)
revert MaxNetDhvExposureExceeded();
      emit NetDhvExposureChanged(netDhvExposure[_optionHashes[i]],
_netDhvExposures[i]);
      netDhvExposure[_optionHashes[i]] = _netDhvExposures[i];
   }
```

The emitted event will include the new exposure and the overridden exposure value. However, it will not contain the appropriate **optionHashes** entry. Therefore, the events do not provide adequate data transparency.

Recommended mitigation

Emit the **_optionHashes[i]** parameter.

Team response

Fixed.

Mitigation review

Fixed correctly using a new indexed event parameter.

TRST-L-5 OptionRegistry could emit a wrong event when issuing tokens

- **Category:** Event emission issues
- **Source:** OptionRegistry.sol
- **Status:** Fixed

Description

The OptionRegistry's *issue()* function emits an event.

```
emit OptionTokenCreated(series);
```

However, the event is misleading. An option token will only be created if it had not existed prior to the call.

```
// check for an opyn oToken if it doesn't exist deploy it
address series = OpynInteractions.getOrDeployOtoken(
   addressBook.getOtokenFactory(),
   optionSeries.collateral,
   optionSeries.underlying,
   optionSeries.strikeAsset,
   formattedStrike,
   optionSeries.expiration,
   optionSeries.isPut
);
```

Recommended mitigation

Check if the **series** has existed prior to the *issue()* call.

Team response

Fixed.

Mitigation review

An early return ensures the event will only be emitted when the oToken is created.

Additional recommendations

Rename functions

The function below is now used to migrate oTokens from the exchange to the handler for OTC orders.

```
/**
 * @notice migrate otokens held by this address to a new option
exchange
 * @param newOptionExchange the option exchange to migrate to
 * @param otokens the otoken addresses to transfer
 */
function migrateOtokens(address newOptionExchange, address[] memory
otokens) external {
```

We recommend updating the variable name and documented to reflect that migration can used to transfer oTokens both to new exchanges and to handlers.

Add safety validations to token migration

The Rysk multisig can migrate tokens through the described function.

```
function migrateOtokens(address optionExchange, address[] memory
otokens) external {
    _onlyGovernor();
    uint256 len = otokens.length;
    for (uint256 i = 0; i < len; i++) {
        if (OtokenInterface(otokens[i]).underlyingAsset() !=
underlyingAsset) {
            revert CustomErrors.NonWhitelistedOtoken();
        }
        uint256 balance = ERC20(otokens[i]).balanceOf(address(this));
        SafeTransferLib.safeTransfer(ERC20(otokens[i]), optionExchange,
balance);
    }
}
```

Note that the function makes sure the **underlyingAsset** is in line with the contract-specific asset. However, it should also check that the new optionExchange's **underlyingAsset** is the same as the oToken **underlyingAsset**. This would protect against potential costly mistakes.

EphemeralDelta should be tracked correctly

Finding L-2 described a location in code where delta is miscalculated. It is also ignored in the pre-upgrade code below:

```
liquidityPool.handlerWriteOption(
    order.optionSeries,
    order.seriesAddress,
    order.amount,
    getOptionRegistry(),
    convertedPrem,
    0, // delta is not used in the liquidityPool unless the oracle
implementation is used, so can be set to 0
    msg.sender
);
```

It is generally not recommended to store state that cannot be relied on. It would be best to either remove the variable, or correct all usages of it to keep it accurate. An incorrectly-tracked integer variable may cause unexpected overflows, impairing the functionality of the contract.

Improve documentation

In the introduction to tenor-based parameters here, it describes:

"Function `initializeTenorParams()` will be added, giving governance the power to overhaul all tenors at once. It will be called in the constructor."

In fact, the function is not called in the constructor. It is called separately.

Optimize gas through early exit

In _getSlippageMultiplier(), if gradient is zero there's nothing to calculate.

```
if (slippageGradient == 0) {
    slippageMultiplier = ONE_SCALE;
    return slippageMultiplier;
}
```

However, it still performs the interpolation algorithm, which is unnecessary.

```
uint256 deltaBandIndex = (uint256(_optionDelta.abs()) * 100) /
deltaBandWidth;
(uint16 tenorIndex, int256 remainder) =
_getTenorIndex(_optionSeries.expiration);
if (_optionDelta < 0) {
    modifiedSlippageGradient = slippageGradient.mul(
        _interpolateSlippageGradient(tenorIndex, remainder, true,
deltaBandIndex)
    );
} else {
    modifiedSlippageGradient = slippageGradient.mul(
        _interpolateSlippageGradient(tenorIndex, remainder, false,
deltaBandIndex)
```

```
    );
}
if (slippageGradient == 0) {
    slippageMultiplier = ONE_SCALE;
    return slippageMultiplier;
}
```

Consider moving the check to the top of the function.

Calculating length outside of loop saves gas

There are several instances in the code where the BeyondPricer loops over arrays, while the condition compares against the dynamically-fetched length value.

```
for (uint256 j = 0; j <
_tenorPricingParams[i].callSlippageGradientMultipliers.length; j++) {
    // arrays must be same length so can check all in same loop
    // ensure no multiplier is less than 1 due to human error.
    if (
```

```
for (uint256 i = 0; i < _callSlippageGradientMultipliers.length; i++)
{
    // arrays must be same length so can check both in same loop
    // ensure no multiplier is less than 1 due to human error.
    if (
```

```
for (uint256 i = 0; i < _callSpreadCollateralMultipliers.length; i++)
{
    // arrays must be same length so can check both in same loop
    // ensure no multiplier is less than 1 due to human error.
```

It is highly recommended to cache the length in a local variable, as it saves an MLOAD instruction for each loop iteration.

BeyondPricer configuration is fragile

There are various setters for parameters that affect option pricing in BeyondPricer. The configuration is considered *live* at all times. If the manager intends to change several of the parameters are once, like the **lowDeltaSellOptionFlatIV** and **lowDeltaThreshold**, users can receive an unintended parameter set between configuration changes. For this reason, it is advisable to have a pause switch for the Pricer. Moreover, the configuration can easily confuse between an unset value and zero. Only in the first case it is desirable to revert pricing. A **lowDeltaSellOptionFlatIV** of zero will be caught in the OptionsCompute library.

Centralization risks

Migration functions can be abused

The *migrateOtokens()* function in OptionExchange and AlphaOptionHandler are naturally a centralization concern, as they could be used to exfiltrate tokens to arbitrary addresses. This would diminish the $-value of the LP.

Pricing parameters can be abused

Pricing is done at the multisig's sole discretion. In theory, pricing could be manipulated to benefit the LP, or conversely the users, at different points in time. Users are urged to be aware of pricing changes and see that they are satisfied with the parameter set offered.