



METATRUST






Draft
Security Assessment for
**37-2021-10-tempus
(10K-FLP) (1Positive-
WOI)**

July 23, 2023






Executive Summary

Overview			
Project Name	37-2021-10-tempus (1OK-FLP) (1Positive-WOI)		
Codebase URL	https://github.com/code-423n4/2021-10-tempus		
Scan Engine	AI Analyzer		
Scan Time	2023/07/23 17:07:53		
Commit Id	df5b441		

Total			
Critical Issues	0		
High risk Issues	8		
Medium risk Issues	0		
Low risk Issues	0		
Informational Issues	0		

Critical Issues		The issue can cause large economic losses, large-scale data disorder, loss of control of authority management, failure of key functions, or indirectly affect the correct operation of other smart contracts interacting with it.
High Risk Issues		The issue puts a large number of users' sensitive information at risk or is reasonably likely to lead to catastrophic impacts on clients' reputations or serious financial implications for clients and users.
Medium Risk Issues		The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk Issues		The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational Issue		The issue does not pose an immediate risk but is relevant to security best practices or Defence in Depth.



	Critical Issues	0%	0
	High risk Issues	100%	8
	Medium risk Issues	0%	0
	Low risk Issues	0%	0
	Informational Issues	0%	0

Summary of Findings

MetaScan security assessment was performed on **July 23, 2023 17:07:53** on project **37-2021-10-tempus (10K-FLP) (1Positive-WOI)** with the repository <https://github.com/code-423n4/2021-10-tempus> on branch **default branch**. The assessment was carried out by scanning the project's codebase using the scan engine **AI Analyzer**. There are in total **8** vulnerabilities / security risks discovered during the scanning session, among which **0** critical vulnerabilities, **8** high risk vulnerabilities, **0** medium risk vulnerabilities, **0** low risk vulnerabilities, **0** informational issues.

ID	Description	Severity
MSA-001	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-002	MWE-209: Wrong Order for Interest or ExchangeRate	High risk
MSA-003	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-004	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-005	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-006	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-007	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-008	MWE-200: Insecure LP Token Value Calculation	High risk



Findings

Critical (0)

No Critical vulnerabilities found here

High risk (8)

1. MWE-200: Insecure LP Token Value Calculation

 High risk Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

contracts/TempusController.sol #337-367

```
337     function _provideLiquidity(  
338         address sender,  
339         IVault vault,  
340         bytes32 poolId,  
341         IERC20[] memory ammTokens,  
342         uint256[] memory ammBalances,  
343         uint256 sharesAmount,  
344         address recipient  
345     ) private returns (uint256[] memory) {  
346         uint256[] memory ammLiquidityProvisionAmounts = ammBalances.getLiquidityProvisionSharesAmounts  
347  
348         if (sender != address(this)) {  
349             ammTokens[0].safeTransferFrom(sender, address(this), ammLiquidityProvisionAmounts[0]);  
350             ammTokens[1].safeTransferFrom(sender, address(this), ammLiquidityProvisionAmounts[1]);  
351         }  
352  
353         ammTokens[0].safeIncreaseAllowance(address(vault), ammLiquidityProvisionAmounts[0]);  
354         ammTokens[1].safeIncreaseAllowance(address(vault), ammLiquidityProvisionAmounts[1]);  
355  
356         IVault.JoinPoolRequest memory request = IVault.JoinPoolRequest({  
357             assets: ammTokens,  
358             maxAmountsIn: ammLiquidityProvisionAmounts,  
359             userData: abi.encode(uint8(ITempusAMM.JoinKind.EXACT_TOKENS_IN_FOR_BPT_OUT), ammLiquidityPro  
360             fromInternalBalance: false  
361         });  
362  
363         // Provide TPS/TYS liquidity to TempusAMM  
364         vault.joinPool(poolId, address(this), recipient, request);  
365  
366         return ammLiquidityProvisionAmounts;  
367     }
```



contracts/TempusController.sol #304-335

```
304     function _depositAndProvideLiquidity(  
305         ITempusAMM tempusAMM,  
306         uint256 tokenAmount,  
307         bool isBackingToken  
308     ) private {  
309         (  
310             IVault vault,  
311             bytes32 poolId,  
312             IERC20[] memory ammTokens,  
313             uint256[] memory ammBalances  
314         ) = _getAMMDetailsAndEnsureInitialized(tempusAMM);  
315  
316         uint256 mintedShares = _deposit(tempusAMM.tempusPool(), tokenAmount, isBackingToken);  
317  
318         uint256[] memory sharesUsed = _provideLiquidity(  
319             address(this),  
320             vault,  
321             poolId,  
322             ammTokens,  
323             ammBalances,  
324             mintedShares,  
325             msg.sender  
326         );  
327  
328         // Send remaining Shares to user  
329         if (sharesUsed[0] < mintedShares) {  
330             ammTokens[0].safeTransfer(msg.sender, mintedShares - sharesUsed[0]);  
331         }  
332         if (sharesUsed[1] < mintedShares) {  
333             ammTokens[1].safeTransfer(msg.sender, mintedShares - sharesUsed[1]);  
334         }  
335     }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

2. MWE-209: Wrong Order for Interest or ExchangeRate

 High risk Security Analyzer

Update of interest or exchange rate should be executed before calculating new balance, share, stake, loan or fee.

File(s) Affected



contracts/TempusPool.sol #257-282

```
257     function burnShares(  
258         address from,  
259         uint256 principalAmount,  
260         uint256 yieldAmount  
261     )  
262     internal  
263     returns (  
264         uint256 redeemedYieldTokens,  
265         uint256 fee,  
266         uint256 interestRate  
267     )  
268     {  
269         require(IERC20(address(principalShare)).balanceOf(from) >= principalAmount, "Insufficient principal");  
270         require(IERC20(address(yieldShare)).balanceOf(from) >= yieldAmount, "Insufficient yields.");  
271  
272         // Redeeming prior to maturity is only allowed in equal amounts.  
273         require(matured || (principalAmount == yieldAmount), "Inequal redemption not allowed before maturity");  
274  
275         // Burn the appropriate shares  
276         PrincipalShare(address(principalShare)).burnFrom(from, principalAmount);  
277         YieldShare(address(yieldShare)).burnFrom(from, yieldAmount);  
278  
279         uint256 currentRate = updateInterestRate();  
280         (redeemedYieldTokens, , fee, interestRate) = getRedemptionAmounts(principalAmount, yieldAmount, currentRate);  
281         totalFees += fee;  
282     }
```

Recommendation

Check the business logic and move the statements about updating exchange rate or interest forward.

3. MWE-200: Insecure LP Token Value Calculation

 High risk Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

contracts/TempusPool.sol #284-332

```
284     function getRedemptionAmounts(  
285         uint256 principalAmount,  
286         uint256 yieldAmount,  
287         uint256 currentRate  
288     )  
289     private  
290     view  
291     returns (  
292         uint256 redeemableYieldTokens,  
293         uint256 redeemableBackingTokens,  
294         uint256 redeemFeeAmount,  
295         uint256 interestRate  
296     )  
297     {  
298         interestRate = effectiveRate(currentRate);  
299  
300         if (interestRate < initialInterestRate) {  
301             redeemableBackingTokens = (principalAmount * interestRate) / initialInterestRate;  
302         } else {  
303             uint256 rateDiff = interestRate - initialInterestRate;  
304             // this is expressed in percent with exchangeRate precision  
305             uint256 yieldPercent = rateDiff.divfV(initialInterestRate, exchangeRateONE);  
306             uint256 redeemAmountFromYieldShares = yieldAmount.mulfV(yieldPercent, exchangeRateONE);  
307  
308             // TODO: Scale based on number of decimals for tokens  
309             redeemableBackingTokens = principalAmount + redeemAmountFromYieldShares;  
310             // after maturity, all additional yield is being collected as fee  
311             if (matured && currentRate > interestRate) {  
312                 uint256 additionalYieldRate = currentRate - interestRate;  
313                 uint256 feeBackingAmount = yieldAmount.mulfV(  
314                     additionalYieldRate.mulfV(initialInterestRate, exchangeRateONE),  
315                     exchangeRateONE  
316                 );  
317                 redeemFeeAmount = numYieldTokensPerAsset(feeBackingAmount, currentRate);  
318             }  
319         }  
320     }  
321  
322     redeemableYieldTokens = numYieldTokensPerAsset(redeemableBackingTokens, currentRate);  
323  
324     uint256 redeemFeePercent = matured ? feesConfig.matureRedeemPercent : feesConfig.earlyRedeemPercent;  
325     if (redeemFeePercent != 0) {  
326         uint256 regularRedeemFee = redeemableYieldTokens.mulfV(redeemFeePercent, yieldBearingONE);  
327         redeemableYieldTokens -= regularRedeemFee;  
328         redeemFeeAmount += regularRedeemFee;  
329  
330         redeemableBackingTokens = numAssetsPerYieldToken(redeemableYieldTokens, currentRate);  
331     }  
332 }
```


contracts/TempusPool.sol #257-282

```
257     function burnShares(  
258         address from,  
259         uint256 principalAmount,  
260         uint256 yieldAmount  
261     )  
262     internal  
263     returns (  
264         uint256 redeemedYieldTokens,  
265         uint256 fee,  
266         uint256 interestRate  
267     )  
268     {  
269         require(IERC20(address(principalShare)).balanceOf(from) >= principalAmount, "Insufficient principal");  
270         require(IERC20(address(yieldShare)).balanceOf(from) >= yieldAmount, "Insufficient yields.");  
271  
272         // Redeeming prior to maturity is only allowed in equal amounts.  
273         require(matured || (principalAmount == yieldAmount), "Inequal redemption not allowed before maturity");  
274  
275         // Burn the appropriate shares  
276         PrincipalShare(address(principalShare)).burnFrom(from, principalAmount);  
277         YieldShare(address(yieldShare)).burnFrom(from, yieldAmount);  
278  
279         uint256 currentRate = updateInterestRate();  
280         (redeemedYieldTokens, fee, interestRate) = getRedemptionAmounts(principalAmount, yieldAmount, currentRate);  
281         totalFees += fee;  
282     }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to calculate LP token value/price.

4. MWE-200: Insecure LP Token Value Calculation



High risk



Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

contracts/TempusPool.sol #370-380

```
370     function estimatedYield(uint256 yieldCurrent) private view returns (uint256) {
371         if (matured) {
372             return yieldCurrent;
373         }
374         uint256 currentTime = block.timestamp;
375         uint256 timeToMaturity = (maturityTime > currentTime) ? (maturityTime - currentTime) : 0;
376         uint256 poolDuration = maturityTime - startTime;
377         uint256 timeLeft = timeToMaturity.divfV(poolDuration, exchangeRateONE);
378
379         return yieldCurrent + timeLeft.mulfV(initialEstimatedYield, exchangeRateONE);
380     }
```

contracts/TempusPool.sol #396-403

```
396     function pricePerPrincipalShare(uint256 currYield, uint256 estYield) private view returns (uint256)
397     // in case we have estimate for negative yield
398     if (estYield < exchangeRateONE) {
399         return interestRateToSharePrice(currYield);
400     }
401     uint256 principalPrice = currYield.divfV(estYield, exchangeRateONE);
402     return interestRateToSharePrice(principalPrice);
403 }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

5. MWE-200: Insecure LP Token Value Calculation



High risk



Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

contracts/TempusPool.sol #384-392

```
384     function pricePerYieldShare(uint256 currYield, uint256 estYield) private view returns (uint256) {
385         uint one = exchangeRateONE;
386         // in case we have estimate for negative yield
387         if (estYield < one) {
388             return uint256(0);
389         }
390         uint256 yieldPrice = (estYield - one).mulfV(currYield, one).divfV(estYield, one);
391         return interestRateToSharePrice(yieldPrice);
392     }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

6. MWE-200: Insecure LP Token Value Calculation



High risk



Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected



contracts/mocks/compound/CTokenMock.sol #48-68

```
48     function mintFresh(address minter, uint mintAmount) internal returns (uint errorCode, uint actualMin
49         uint err = comptroller.mintAllowed(address(this), minter, mintAmount);
50         require(err == 0, "mint is not allowed");
51
52         uint exchangeRate = exchangeRateStored(); // exchangeRate has variable decimal precision
53
54         /*
55          * We call `doTransferIn` for the minter and the mintAmount.
56          * Note: The cToken must handle variations between ERC-20 and ETH underlying.
57          * `doTransferIn` reverts if anything goes wrong, since we can't be sure if
58          * side-effects occurred. The function returns the amount actually transferred,
59          * in case of a fee. On success, the cToken holds an additional `actualMintAmount`
60          * of cash.
61          */
62         actualMintAmount = doTransferIn(minter, mintAmount); // 18 decimal precision
63
64         // exchange rate precision: 18 - 8 + Underlying Token Decimals
65         uint mintTokens = (actualMintAmount * 1e18) / exchangeRate; // (18 + 18) - 28 = 8 decimal precis
66         _mint(minter, mintTokens);
67         errorCode = 0;
68     }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to caculate LP token value/price.

7. MWE-200: Insecure LP Token Value Calculation

 High risk Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected



contracts/amm/TempusAMM.sol #155-171

```
155     function getExpectedReturnGivenIn(uint256 amount, bool yieldShareIn) public view returns (uint256)
156     {
157         (, uint256[] memory balances, ) = getVault().getPoolTokens(getPoolId());
158         (uint256 currentAmp, ) = _getAmplificationParameter();
159         (IPoolShare tokenIn, IPoolShare tokenOut) = yieldShareIn
160             ? (tempusPool.yieldShare(), tempusPool.principalShare())
161             : (tempusPool.principalShare(), tempusPool.yieldShare());
162         (uint256 indexIn, uint256 indexOut) = address(tokenIn) == address(_token0) ? (0, 1) : (1, 0);
163
164         amount = _subtractSwapFeeAmount(amount);
165         balances.mul(_getTokenRatesStored(), _TEMPUS_SHARE_PRECISION);
166         uint256 rateAdjustedSwapAmount = (amount * tokenIn.getPricePerFullShareStored()) / _TEMPUS_SHARE_PRECISION;
167
168         uint256 amountOut = StableMath._calcOutGivenIn(currentAmp, balances, indexIn, indexOut, rateAdjustedSwapAmount);
169         amountOut = (amountOut * _TEMPUS_SHARE_PRECISION) / tokenOut.getPricePerFullShareStored();
170
171         return amountOut;
172     }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to calculate LP token value/price.

8. MWE-200: Insecure LP Token Value Calculation

 High risk Security Analyzer

Liquidity token value/price can be manipulated to cause flashloan attacks.

File(s) Affected

contracts/amm/TempusAMM.sol #686-691

```
686     function _getTokenRatesStored() private view returns (uint256[] memory) {
687         uint256[] memory rates = new uint256[](_TOTAL_TOKENS);
688         rates[0] = _token0.getPricePerFullShareStored();
689         rates[1] = _token1.getPricePerFullShareStored();
690         return rates;
691     }
```

contracts/amm/TempusAMM.sol #258-277

```
258     function getExpectedLPTokensForTokensIn(uint256[] memory amountsIn) external view returns (uint256)
259         (, uint256[] memory balances, ) = getVault().getPoolTokens(getPoolId());
260
261     uint256[] memory tokenRates = _getTokenRatesStored();
262     balances.mul(tokenRates, _TEMPUS_SHARE_PRECISION);
263     amountsIn.mul(tokenRates, _TEMPUS_SHARE_PRECISION);
264
265     (uint256 currentAmp, ) = _getAmplificationParameter();
266
267     return
268         (balances[0] == 0)
269             ? StableMath._calculateInvariant(currentAmp, amountsIn, true)
270             : StableMath._calcBptOutGivenExactTokensIn(
271                 currentAmp,
272                 balances,
273                 amountsIn,
274                 totalSupply(),
275                 getSwapFeePercentage()
276             );
277 }
```

Recommendation

Do not use AMM pool or custom liquidity calculation to calculate LP token value/price.

Medium risk (0)

No Medium risk vulnerabilities found here

Low risk (0)

No Low risk vulnerabilities found here

Informational (0)

No Informational vulnerabilities found here

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without MetaTrust's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts MetaTrust to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. MetaTrust's position is that each company and individual are responsible for their own due diligence and continuous security. MetaTrust's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by MetaTrust is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS 37-2021-10-tempus (1OK-FLP) (1Positive-WOI) Security Assessment AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW,

MetaTrust HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, MetaTrust SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, MetaTrust MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, MetaTrust PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER MetaTrust NOR ANY OF MetaTrust'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. MetaTrust WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT MetaTrust'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING 37-2021-10-tempus (1OK-FLP) (1Positive-WOI) Security Assessment MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST MetaTrust WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF MetaTrust CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST MetaTrust WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.