



METATRUST

Draft
Security Assessment for
**23-2021-08-notional
(1Positive-SP)
(1Negative-FLP)**






July 23, 2023

Executive Summary

Overview			
Project Name	23-2021-08-notional (1Positive-SP) (1Negative-FLP)		Critical Issues
Codebase URL	https://github.com/code-423n4/2021-08-notional		High Risk Issues
Scan Engine	AI Analyzer		Medium Risk Issues
Scan Time	2023/07/23 22:07:29		Low Risk Issues
Commit Id	8368d59		Informational Issue

Total	
Critical Issues	0
High risk Issues	4
Medium risk Issues	0
Low risk Issues	0
Informational Issues	0



	Critical Issues	0%	0
	High risk Issues	100%	4
	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 22:07:29** on project **23-2021-08-notional (1Positive-SP) (1Negative-FLP)** with the repository **<https://github.com/code-423n4/2021-08-notional>** 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 **4** vulnerabilities / security risks discovered during the scanning session, among which **0** critical vulnerabilities, **4** high risk vulnerabilities, **0** medium risk vulnerabilities, **0** low risk vulnerabilities, **0** informational issues.

ID	Description	Severity
MSA-001	MWE-206: No Slippage Limit Check	High risk
MSA-002	MWE-206: No Slippage Limit Check	High risk
MSA-003	MWE-206: No Slippage Limit Check	High risk
MSA-004	MWE-206: No Slippage Limit Check	High risk



Findings

Critical (0)

No Critical vulnerabilities found here

High risk (4)

1. MWE-206: No Slippage Limit Check

 High risk Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

contracts/mocks/MockMarket.sol #127-141

```
127     function removeLiquidity(MarketParameters memory marketState, int256 tokensToRemove)
128     public
129     pure
130     returns (
131         MarketParameters memory,
132         int256,
133         int256
134     )
135     {
136         (int256 assetCash, int256 fCash) = marketState.removeLiquidity(tokensToRemove);
137
138         assert(assetCash >= 0);
139         assert(fCash >= 0);
140         return (marketState, assetCash, fCash);
141     }
```

contracts/internal/liquidation/LiquidateCurrency.sol #440-502


```
440     function _withdrawCollateralLiquidityTokens(
441         PortfolioState memory portfolioState,
442         LiquidationFactors memory factors,
443         uint256 blockTime,
444         int256 collateralToWithdraw
445     ) internal view returns (int256) {
446         require(portfolioState.newAssets.length == 0); // dev: new assets in portfolio
447         factors.markets = new MarketParameters[] (factors.cashGroup.maxMarketIndex);
448
449         for (uint256 i; i < portfolioState.storedAssets.length; i++) {
450             PortfolioAsset memory asset = portfolioState.storedAssets[i];
451             if (asset.storageState == AssetStorageState.Delete) continue;
452             if (
453                 !AssetHandler.isLiquidityToken(asset.assetType) ||
454                 asset.currencyId != factors.cashGroup.currencyId
455             ) continue;
456
457             uint256 marketIndex = asset.assetType - 1;
458             // This is set up this way so that we can delay setting storage of markets so that this met
459             // remain a view function
460             factors.cashGroup.loadMarket(
461                 factors.markets[marketIndex - 1],
462                 marketIndex,
463                 true,
464                 blockTime
465             );
466             (int256 cashClaim, int256 fCashClaim) =
467                 asset.getCashClaims(factors.markets[marketIndex - 1]);
468
469             if (cashClaim <= collateralToWithdraw) {
470                 // The additional cash is insufficient to cover asset amount required so we just remove
471                 portfolioState.deleteAsset(i);
472                 factors.markets[marketIndex - 1].removeLiquidity(asset.notional);
473
474                 // overflow checked above
475                 collateralToWithdraw = collateralToWithdraw - cashClaim;
476             } else {
477                 // Otherwise remove a proportional amount of liquidity tokens to cover the amount rema
478                 // NOTE: dust can accrue when withdrawing liquidity at this point
479                 int256 tokensToRemove = asset.notional.mul(collateralToWithdraw).div(cashClaim);
480                 (cashClaim, fCashClaim) = factors.markets[marketIndex - 1].removeLiquidity(
481                     tokensToRemove
482                 );
483
484                 // Remove liquidity token balance
485                 portfolioState.storedAssets[i].notional = asset.notional.subNoNeg(tokensToRemove);
486                 portfolioState.storedAssets[i].storageState = AssetStorageState.Update;
487                 collateralToWithdraw = 0;
488             }
489
490             // Add the netfCash asset to the portfolio since we've withdrawn the liquidity tokens
491             portfolioState.addAsset(
492                 factors.cashGroup.currencyId,
493                 asset.maturity,
494                 Constants.FCASH_ASSET_TYPE,
495                 fCashClaim
496             );
497         }
498     }
```

```
497
498         if (collateralToWithdraw == 0) return 0;
499     }
500
501     return collateralToWithdraw;
502 }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

2. MWE-206: No Slippage Limit Check

 High risk Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

contracts/internal/liquidation/LiquidateCurrency.sol #334-416

```
334     function _withdrawLocalLiquidityTokens(
335         PortfolioState memory portfolioState,
336         LiquidationFactors memory factors,
337         uint256 blockTime,
338         int256 assetAmountRemaining
339     ) internal view returns (WithdrawFactors memory, int256) {
340         require(portfolioState.newAssets.length == 0); // dev: new assets in portfolio
341         factors.markets = new MarketParameters[] (factors.cashGroup.maxMarketIndex);
342         // Do this to deal with stack issues
343         WithdrawFactors memory w;
344
345         for (uint256 i; i < portfolioState.storedAssets.length; i++) {
346             PortfolioAsset memory asset = portfolioState.storedAssets[i];
347             if (asset.storageState == AssetStorageState.Delete) continue;
348             if (
349                 !AssetHandler.isLiquidityToken(asset.assetType) ||
350                 asset.currencyId != factors.cashGroup.currencyId
351             ) continue;
352
353             uint256 marketIndex = asset.assetType - 1;
354             // This is set up this way so that we can delay setting storage of markets so that this me
355             // remain a view function
356             factors.cashGroup.loadMarket(
357                 factors.markets[marketIndex - 1],
358                 marketIndex,
359                 true,
360                 blockTime
361             );
362
363             // NOTE: we do not give any credit to the haircut fCash in this procedure but it will end u
364             // additional collateral value back into the account. It's probably too complex to deal wit
365             // we will just leave it as such.
366             (w.assetCash, w.fCash) = asset.getCashClaims(factors.markets[marketIndex - 1]);
367             _calculateNetCashIncreaseAndIncentivePaid(factors, w, asset.assetType);
368
369             // (netCashToAccount <= assetAmountRemaining)
370             if (w.netCashIncrease.subNoNeg(w.incentivePaid) <= assetAmountRemaining) {
371                 // The additional cash is insufficient to cover asset amount required so we just remove
372                 portfolioState.deleteAsset(i);
373                 factors.markets[marketIndex - 1].removeLiquidity(asset.notional);
374
375                 // assetAmountRemaining = assetAmountRemaining - netCashToAccount
376                 // netCashToAccount = netCashIncrease - incentivePaid
377                 // overflow checked above
378                 assetAmountRemaining =
379                     assetAmountRemaining -
380                     w.netCashIncrease.sub(w.incentivePaid);
381             } else {
382                 // Otherwise remove a proportional amount of liquidity tokens to cover the amount rema
383                 int256 tokensToRemove =
384                     asset.notional.mul(assetAmountRemaining).div(
385                         w.netCashIncrease.subNoNeg(w.incentivePaid)
386                     );
387
388                 (w.assetCash, w.fCash) = factors.markets[marketIndex - 1].removeLiquidity(
389                     tokensToRemove
390                 );
391             }
392         }
393         return (w, assetAmountRemaining);
394     }
```

```
391 // Recalculate net cash increase and incentive paid. w.assetCash is different because ;
392 // remove asset cash
393 _calculateNetCashIncreaseAndIncentivePaid(factors, w, asset.assetType);
394
395 // Remove liquidity token balance
396 portfolioState.storedAssets[i].notional = asset.notional.subNoNeg(tokensToRemove);
397 portfolioState.storedAssets[i].storageState = AssetStorageState.Update;
398 assetAmountRemaining = 0;
399 }
400
401 w.totalIncentivePaid = w.totalIncentivePaid.add(w.incentivePaid);
402 w.totalCashClaim = w.totalCashClaim.add(w.assetCash);
403
404 // Add the netfCash asset to the portfolio since we've withdrawn the liquidity tokens
405 portfolioState.addAsset(
406     factors.cashGroup.currencyId,
407     asset.maturity,
408     Constants.FCASH_ASSET_TYPE,
409     w.fCash
410 );
411
412 if (assetAmountRemaining == 0) break;
413 }
414
415 return (w, assetAmountRemaining);
416 }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

3. MWE-206: No Slippage Limit Check



High risk



Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

contracts/internal/liquidation/LiquidateCurrency.sol #334-416

```
334     function _withdrawLocalLiquidityTokens(  
335         PortfolioState memory portfolioState,  
336         LiquidationFactors memory factors,  
337         uint256 blockTime,  
338         int256 assetAmountRemaining  
339     ) internal view returns (WithdrawFactors memory, int256) {  
340         require(portfolioState.newAssets.length == 0); // dev: new assets in portfolio  
341         factors.markets = new MarketParameters[] (factors.cashGroup.maxMarketIndex);  
342         // Do this to deal with stack issues  
343         WithdrawFactors memory w;  
344  
345         for (uint256 i; i < portfolioState.storedAssets.length; i++) {  
346             PortfolioAsset memory asset = portfolioState.storedAssets[i];  
347             if (asset.storageState == AssetStorageState.Delete) continue;  
348             if (  
349                 !AssetHandler.isLiquidityToken(asset.assetType) ||  
350                 asset.currencyId != factors.cashGroup.currencyId  
351             ) continue;  
352  
353             uint256 marketIndex = asset.assetType - 1;  
354             // This is set up this way so that we can delay setting storage of markets so that this met  
355             // remain a view function  
356             factors.cashGroup.loadMarket(  
357                 factors.markets[marketIndex - 1],  
358                 marketIndex,  
359                 true,  
360                 blockTime  
361             );  
362  
363             // NOTE: we do not give any credit to the haircut fCash in this procedure but it will end u  
364             // additional collateral value back into the account. It's probably too complex to deal wit  
365             // we will just leave it as such.  
366             (w.assetCash, w.fCash) = asset.getCashClaims(factors.markets[marketIndex - 1]);  
367             _calculateNetCashIncreaseAndIncentivePaid(factors, w, asset.assetType);  
368  
369             // (netCashToAccount <= assetAmountRemaining)  
370             if (w.netCashIncrease.subNoNeg(w.incentivePaid) <= assetAmountRemaining) {  
371                 // The additional cash is insufficient to cover asset amount required so we just remove  
372                 portfolioState.deleteAsset(i);  
373                 factors.markets[marketIndex - 1].removeLiquidity(asset.notional);  
374  
375                 // assetAmountRemaining = assetAmountRemaining - netCashToAccount  
376                 // netCashToAccount = netCashIncrease - incentivePaid  
377                 // overflow checked above  
378                 assetAmountRemaining =  
379                     assetAmountRemaining -  
380                     w.netCashIncrease.sub(w.incentivePaid);  
381             } else {  
382                 // Otherwise remove a proportional amount of liquidity tokens to cover the amount rema  
383                 int256 tokensToRemove =  
384                     asset.notional.mul(assetAmountRemaining).div(  
385                         w.netCashIncrease.subNoNeg(w.incentivePaid)  
386                     );  
387  
388                 (w.assetCash, w.fCash) = factors.markets[marketIndex - 1].removeLiquidity(  
389                     tokensToRemove  
390                 );  
391             }  
392         }  
393         return (w, assetAmountRemaining);  
394     }
```

```
391         // Recalculate net cash increase and incentive paid. w.assetCash is different because ;
392         // remove asset cash
393         _calculateNetCashIncreaseAndIncentivePaid(factors, w, asset.assetType);
394
395         // Remove liquidity token balance
396         portfolioState.storedAssets[i].notional = asset.notional.subNoNeg(tokensToRemove);
397         portfolioState.storedAssets[i].storageState = AssetStorageState.Update;
398         assetAmountRemaining = 0;
399     }
400
401     w.totalIncentivePaid = w.totalIncentivePaid.add(w.incentivePaid);
402     w.totalCashClaim = w.totalCashClaim.add(w.assetCash);
403
404     // Add the netfCash asset to the portfolio since we've withdrawn the liquidity tokens
405     portfolioState.addAsset(
406         factors.cashGroup.currencyId,
407         asset.maturity,
408         Constants.FCASH_ASSET_TYPE,
409         w.fCash
410     );
411
412     if (assetAmountRemaining == 0) break;
413 }
414
415 return (w, assetAmountRemaining);
416 }
```

contracts/internal/liquidation/LiquidateCurrency.sol #50-130

```
50     function liquidateLocalCurrency(  
51         uint256 localCurrency,  
52         uint96 maxNTokenLiquidation,  
53         uint256 blockTime,  
54         BalanceState memory balanceState,  
55         LiquidationFactors memory factors,  
56         PortfolioState memory portfolio  
57     ) internal view returns (int256) {  
58         require(factors.localAssetAvailable < 0, "No local debt");  
59  
60         int256 assetBenefitRequired =  
61             factors.cashGroup.assetRate.convertFromUnderlying(  
62                 factors  
63                     .localETHRate  
64                     .convertETHTo(factors.netETHValue.neg())  
65                     .mul(Constants.PERCENTAGE_DECIMALS)  
66                     .div(factors.localETHRate.buffer)  
67             );  
68  
69         int256 netAssetCashFromLiquidator;  
70  
71         if (_hasLiquidityTokens(portfolio.storedAssets, localCurrency)) {  
72             WithdrawFactors memory w;  
73             (w, assetBenefitRequired) = _withdrawLocalLiquidityTokens(  
74                 portfolio,  
75                 factors,  
76                 blockTime,  
77                 assetBenefitRequired  
78             );  
79             netAssetCashFromLiquidator = w.totalIncentivePaid.neg();  
80             balanceState.netCashChange = w.totalCashClaim.sub(w.totalIncentivePaid);  
81         }  
82  
83         if (factors.nTokenHaircutAssetValue > 0) {  
84             int256 nTokensToLiquidate;  
85             {  
86                 // This will not underflow, checked when saving parameters  
87                 int256 haircutDiff =  
88                     int256(  
89                         uint8(factors.nTokenParameters[Constants.LIQUIDATION_HAIRCUT_PERCENTAGE]) -  
90                         uint8(factors.nTokenParameters[Constants.PV_HAIRCUT_PERCENTAGE])  
91                     ) * Constants.PERCENTAGE_DECIMALS;  
92  
93                 // fullNTokenPV = haircutTokenPV / haircutPercentage  
94                 // benefitGained = nTokensToLiquidate * (liquidatedPV - freeCollateralPV)  
95                 // benefitGained = nTokensToLiquidate * (fullNTokenPV * liquidatedPV - fullNTokenPV * pv  
96                 // benefitGained = nTokensToLiquidate * fullNTokenPV * (liquidatedPV - pvHaircut) / tota  
97                 // benefitGained = nTokensToLiquidate * (haircutTokenPV / haircutPercentage) * (liquidat  
98                 // benefitGained = nTokensToLiquidate * haircutTokenPV * (liquidationHaircut - pvHaircut  
99                 // nTokensToLiquidate = (benefitGained * totalBalance * haircutPercentage) / (haircutTok  
100                 nTokensToLiquidate = assetBenefitRequired  
101                     .mul(balanceState.storedNTokenBalance)  
102                     .mul(int256(uint8(factors.nTokenParameters[Constants.PV_HAIRCUT_PERCENTAGE])))  
103                     .div(factors.nTokenHaircutAssetValue.mul(haircutDiff));  
104             }  
105  
106             nTokensToLiquidate = LiquidationHelpers.calculateLiquidationAmount(  


```

```
107         nTokensToLiquidate,  
108         balanceState.storedNTokenBalance,  
109         int256(maxNTokenLiquidation)  
110     );  
111     balanceState.netNTokenTransfer = nTokensToLiquidate.neg();  
112  
113     {  
114         // fullNTokenPV = haircutTokenPV / haircutPercentage  
115         // localFromLiquidator = tokensToLiquidate * fullNTokenPV * liquidationHaircut / totalB  
116         // prettier-ignore  
117         int256 localAssetCash =  
118             nTokensToLiquidate  
119                 .mul(int256(uint8(factors.nTokenParameters[Constants.LIQUIDATION_HAIRCUT_PERCENTAGE]  
120                 .mul(factors.nTokenHaircutAssetValue)  
121                 .div(int256(uint8(factors.nTokenParameters[Constants.PV_HAIRCUT_PERCENTAGE]))))  
122                 .div(balanceState.storedNTokenBalance);  
123  
124         balanceState.netCashChange = balanceState.netCashChange.add(localAssetCash);  
125         netAssetCashFromLiquidator = netAssetCashFromLiquidator.add(localAssetCash);  
126     }  
127 }  
128  
129 return netAssetCashFromLiquidator;  
130 }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

4. MWE-206: No Slippage Limit Check

 High risk Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

contracts/internal/liquidation/LiquidateCurrency.sol #440-502

```
440     function _withdrawCollateralLiquidityTokens(
441         PortfolioState memory portfolioState,
442         LiquidationFactors memory factors,
443         uint256 blockTime,
444         int256 collateralToWithdraw
445     ) internal view returns (int256) {
446         require(portfolioState.newAssets.length == 0); // dev: new assets in portfolio
447         factors.markets = new MarketParameters[] (factors.cashGroup.maxMarketIndex);
448
449         for (uint256 i; i < portfolioState.storedAssets.length; i++) {
450             PortfolioAsset memory asset = portfolioState.storedAssets[i];
451             if (asset.storageState == AssetStorageState.Delete) continue;
452             if (
453                 !AssetHandler.isLiquidityToken(asset.assetType) ||
454                 asset.currencyId != factors.cashGroup.currencyId
455             ) continue;
456
457             uint256 marketIndex = asset.assetType - 1;
458             // This is set up this way so that we can delay setting storage of markets so that this met
459             // remain a view function
460             factors.cashGroup.loadMarket(
461                 factors.markets[marketIndex - 1],
462                 marketIndex,
463                 true,
464                 blockTime
465             );
466             (int256 cashClaim, int256 fCashClaim) =
467                 asset.getCashClaims(factors.markets[marketIndex - 1]);
468
469             if (cashClaim <= collateralToWithdraw) {
470                 // The additional cash is insufficient to cover asset amount required so we just remove
471                 portfolioState.deleteAsset(i);
472                 factors.markets[marketIndex - 1].removeLiquidity(asset.notional);
473
474                 // overflow checked above
475                 collateralToWithdraw = collateralToWithdraw - cashClaim;
476             } else {
477                 // Otherwise remove a proportional amount of liquidity tokens to cover the amount rema
478                 // NOTE: dust can accrue when withdrawing liquidity at this point
479                 int256 tokensToRemove = asset.notional.mul(collateralToWithdraw).div(cashClaim);
480                 (cashClaim, fCashClaim) = factors.markets[marketIndex - 1].removeLiquidity(
481                     tokensToRemove
482                 );
483
484                 // Remove liquidity token balance
485                 portfolioState.storedAssets[i].notional = asset.notional.subNoNeg(tokensToRemove);
486                 portfolioState.storedAssets[i].storageState = AssetStorageState.Update;
487                 collateralToWithdraw = 0;
488             }
489
490             // Add the netfCash asset to the portfolio since we've withdrawn the liquidity tokens
491             portfolioState.addAsset(
492                 factors.cashGroup.currencyId,
493                 asset.maturity,
494                 Constants.FCASH_ASSET_TYPE,
495                 fCashClaim
496             );
497         }
498     }
```

```
497
498         if (collateralToWithdraw == 0) return 0;
499     }
500
501     return collateralToWithdraw;
502 }
```

NON-OFFICIAL AUDIT REPORT

NON-OFFICIAL AUDIT REPORT

NON-OFFI

NON-OFFICIAL AUDIT REPORT

NON-OFFICIAL AUDIT REPORT

NON-OFFI

NON-OFFICIAL AUDIT REPORT

NON-OFFICIAL AUDIT REPORT

NON-OFFI

NON-OFFICIAL AUDIT REPORT

NON-OFFICIAL AUDIT REPORT

NON-OFFI

NON-OFFICIAL AUDIT REPORT

NON-OFFICIAL AUDIT REPORT

NON-OFFI

contracts/internal/liquidation/LiquidateCurrency.sol #134-213

```
134     function liquidateCollateralCurrency(  
135         uint128 maxCollateralLiquidation,  
136         uint96 maxNTokenLiquidation,  
137         uint256 blockTime,  
138         BalanceState memory balanceState,  
139         LiquidationFactors memory factors,  
140         PortfolioState memory portfolio  
141     ) internal view returns (int256) {  
142         require(factors.localAssetAvailable < 0, "No local debt");  
143         require(factors.collateralAssetAvailable > 0, "No collateral");  
144  
145         (  
146             int256 requiredCollateralAssetCash,  
147             int256 localAssetCashFromLiquidator,  
148             int256 liquidationDiscount  
149         ) = _calculateCollateralToRaise(factors, int256(maxCollateralLiquidation));  
150  
151         int256 collateralAssetRemaining = requiredCollateralAssetCash;  
152         if (balanceState.storedCashBalance > 0) {  
153             if (balanceState.storedCashBalance > collateralAssetRemaining) {  
154                 balanceState.netCashChange = collateralAssetRemaining.neg();  
155                 collateralAssetRemaining = 0;  
156             } else {  
157                 // Sell off all cash balance and calculate remaining collateral  
158                 balanceState.netCashChange = balanceState.storedCashBalance.neg();  
159                 collateralAssetRemaining = collateralAssetRemaining.sub(  
160                     balanceState.storedCashBalance  
161                 );  
162             }  
163         }  
164  
165         if (  
166             collateralAssetRemaining > 0 &&  
167             _hasLiquidityTokens(portfolio.storedAssets, balanceState.currencyId)  
168         ) {  
169             int256 newCollateralAssetRemaining =  
170                 _withdrawCollateralLiquidityTokens(  
171                     portfolio,  
172                     factors,  
173                     blockTime,  
174                     collateralAssetRemaining  
175                 );  
176  
177             // This is a hack and ugly but there are stack issues in `LiquidateCurrencyAction.liquidate`  
178             // and this is a way to deal with it with the fewest contortions. There are no asset cash t  
179             // so we overload the meaning of the field here to hold the net liquidity token cash change  
180             // going into finalize for the liquidated account's cash balances. This value is not simply  
181             // because the cashClaim value is not stored in the balances and therefore the liquidated a  
182             // debited from their stored cash value.  
183             balanceState.netAssetTransferInternalPrecision = collateralAssetRemaining.sub(  
184                 newCollateralAssetRemaining  
185             );  
186             collateralAssetRemaining = newCollateralAssetRemaining;  
187         }  
188  
189         if (collateralAssetRemaining > 0 && factors.nTokenHaircutAssetValue > 0) {  
190             collateralAssetRemaining = _calculateCollateralNTokenTransfer(  

```

```
191         balanceState,  
192         factors,  
193         collateralAssetRemaining,  
194         int256(maxNTokenLiquidation)  
195     );  
196 }  
197  
198     if (collateralAssetRemaining > 0) {  
199         // If there is any collateral asset remaining then recalculate the localAssetCashFromLiqui  
200         // prettier-ignore  
201         (  
202             /* collateralToRaise */,  
203             localAssetCashFromLiquidator  
204         ) = LiquidationHelpers.calculateLocalToPurchase(  
205             factors,  
206             liquidationDiscount,  
207             requiredCollateralAssetCash.sub(collateralAssetRemaining),  
208             requiredCollateralAssetCash.sub(collateralAssetRemaining)  
209         );  
210     }  
211  
212     return localAssetCashFromLiquidator;  
213 }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

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 23-2021-08-notional (1Positive-SP) (1Negative-FLP) 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 23-2021-08-notional (1Positive-SP) (1Negative-FLP) 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.