METATRUST

Security Assessment for MUFEX

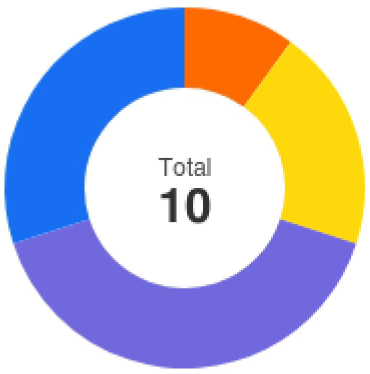
June 8, 2023

Executive Summary

Overview	
Project Name	MUFEX
Codebase URL	-
Scan Engine	Security Analyzer
Scan Time	2023/06/8 16:21:10
Commit Id	-

Total	
Critical Issues	0
High risk Issues	1
Medium risk Issues	2
Low risk Issues	4
Informational Issues	3

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	10%	1
Medium risk Issues	20%	2
Low risk Issues	40%	4
Informational Issues	30%	3

Summary of Findings

MetaScan security assessment was performed on **June 8, 2023 16:21:10** on project **MUFEX** with the repository **MUFEX** on branch **-**. The assessment was carried out by scanning the project's codebase using the scan engine **Security Analyzer**. There are in total **10** vulnerabilities / security risks discovered during the scanning session, among which **0** critical vulnerabilities, **1** high risk vulnerabilities, **2** medium risk vulnerabilities, **4** low risk vulnerabilities, **3** informational issues.

ID	Description	Severity	Alleviation
MSA-001	Inappropriate Handling of Ether Balances in updateZKP Function	High risk	Fixed
MSA-002	Potential DoS when updating ZKP	Medium risk	Acknowledged
MSA-003	Out-of-Bounds Array Assignment in `generalWithdraw` Function	Medium risk	Fixed
MSA-004	Lack of zero address check	Low risk	Fixed
MSA-005	Lack of Access Control	Low risk	Fixed
MSA-006	Gas limitation for the `receive` function	Low risk	Acknowledged
MSA-007	DoS attack when creating a wallet	Low risk	Acknowledged
MSA-008	Gas savings	Informational	Fixed
MSA-009	Unclear error in require logic	Informational	Acknowledged
MSA-010	Potential Repeated Item Inserted into `allGeneralWithdrawnIndex` or `allForceWithdrawnIndex`	Informational	Acknowledged

Findings

Critical (0)

No Critical vulnerabilities found here

High risk (1)

1. Inappropriate Handling of Ether Balances in updateZKP Function — High risk — Security Analyzer

The MainTreasury contract does not appropriately handle Ether balances in the updateZKP function. The function iterates over a list of tokens and checks if the contract has enough balance of each token. However, when it comes to handling Ether, the contract still tries to use the ERC20 balanceOf method, which is inappropriate for Ether as Ether is not an ERC20 token.

File(s) Affected

contracts/MainTreasury.sol #62-62

```
62          uint256 balanceOfThis = IERC20(token).balanceOf(address(this));
```

Recommendation

here are two potential solutions to this issue, depending on the intended functionality of the code: Modify the code to handle Ether balances separately using address(this).balance for the Ether case.

Alleviation Fixed

The development team fixed this issue in commit https://github.com/MUFEX-Exchange/smart-contract/commit/e2091a77d215c97e689bc98eb9232721ed8a26d0

Medium risk (2)

1. Potential DoS when updating ZKP — Medium risk — Security Analyzer

For the version of commit `056df89e788c8e35f03c7a37df3eefbe81ca4127`, on May 30.

The `updateZKP` function requires that `newZkpId` is greater than `zkpId` as shown below:
`solidity function updateZKP( uint64 newZkpId, uint256 newBalanceRoot, uint256 newWithdrawRoot, uint256 newTotalBalance, uint256 newTotalWithdraw ) external override onlyVerifierSet { ... require(newZkpId > zkpId, "old zkp"); ...`

However, what if a `newZkpId` is set to the `type(uint64).max` by mistake, which results in the next update will always fail since `newZkpId > type(uint64).max` returns false.

File(s) Affected

contracts/MainTreasury.sol #64-64

```
64          require(newZkpId > zkpId, "old zkp");
```

Recommendation

Checking if it is an intended design, if not, consider increasing `zkpId` by one per update.

Alleviation Acknowledged

The development team acknowledged this issue.

2. Out-of-Bounds Array Assignment in `generalWithdraw` Function

Medium risk Security Analyzer

In the provided generalWithdraw function, there is an error with the msgs array. The array is initialized with a size of 8 `(new uint256[](8))`, but it tries to assign a value to the 9th element `(msgs[8] = amount;)`. This will cause an out-of-bounds error because arrays in Solidity are 0-indexed, meaning that the index of the last element of an array with size 8 is 7.

File(s) Affected

contracts/MainTreasury.sol #95-104

```
        uint256[] memory msgs = new uint256[](8);
        msgs[0] = zkpId;
        msgs[1] = index;
        msgs[2] = withdrawId;
        msgs[3] = accountId;
        msgs[4] = uint256(uint160(account));
        msgs[5] = uint256(uint160(to));
        msgs[6] = withdrawType;
        msgs[7] = amount;
        uint256 node = MiMC.Hash(msgs);
```

Recommendation

1. If all 9 elements are required, increase the size of the msgs array to 9 during initialization:

`uint256[] memory msgs = new uint256[](9);`

This will create an array with enough space for the 9 elements.

2. If the assignment to the 9th element is not required, simply remove the line msgs[8] = amount;.

Alleviation Fixed

The development team fixed this issue in commit https://github.com/MUFEX-Exchange/smart-contract/commit/123e80f8f0d84d4583be57d320d0278e04c0f99b

Low risk (4)

1. Lack of zero address check

Low risk Security Analyzer

For the version of commit `056df89e788c8e35f03c7a37df3eefbe81ca4127`, on May 30.

Zero addresses assigned to the address type state variables will result in an unexpected result.

Example:
`solidity constructor(address treasury_) { treasury = treasury_; }`

File(s) Affected

Recommendation

Adding zero value check on address type state variables.

Alleviation Fixed

The development team resolved this issue in the commit https://github.com/MUFEX-Exchange/smart-contract/commit/123e80f8f0d84d4583be57d320d0278e04c0f99b

2. Lack of Access Control

Low risk Security Analyzer

For the version of commit `056df89e788c8e35f03c7a37df3eefbe81ca4127`, on May 30.

In the `Verifier` contract, the `submit` function invokes `updateZKP` function of the `mainTreasury` contract.

However, there is no access control in the `submit` function, which results in anyone can submit a `ZKP` and leads to unexpected results.

File(s) Affected

contracts/Verifier.sol #102-148

```
    function submit(
        uint64 zkpId,
        uint256[] memory BeforeAccountTreeRoot,
        uint256[] memory AfterAccountTreeRoot,
        uint256[] memory BeforeCEXAssetsCommitment,
        uint256[] memory AfterCEXAssetsCommitment,
        uint256[2][] memory a, // zk proof
        uint256[2][2][] memory b, // zk proof
        uint256[2][] memory c, // zk proof
        uint256 withdrawMerkelTreeToot,//@audit typo
        uint256 totalBalance,
        uint256 totalWithdraw
    ) public returns (bool r) {//@audit lack access control
        //
        require(BeforeAccountTreeRoot.length == AfterAccountTreeRoot.length,"BeforeAccountTreeRoot.leng
        require(BeforeAccountTreeRoot.length == BeforeCEXAssetsCommitment.length,"BeforeAccountTreeRoot
        require(BeforeAccountTreeRoot.length == AfterCEXAssetsCommitment.length,"BeforeAccountTreeRoot.
        require(BeforeAccountTreeRoot.length == a.length,"BeforeAccountTreeRoot.length != a.length");
        require(BeforeAccountTreeRoot.length == b.length,"BeforeAccountTreeRoot.length != b.length");
        require(BeforeAccountTreeRoot.length == c.length,"BeforeAccountTreeRoot.length != c.length");

        //          after        before
        for (uint256 i = 1; i < BeforeAccountTreeRoot.length; i++) {
            require(BeforeAccountTreeRoot[i] == AfterAccountTreeRoot[i-1],"BeforeAccountTreeRoot[i] !=
            require(BeforeCEXAssetsCommitment[i] == AfterCEXAssetsCommitment[i-1],"BeforeCEXAssetsComm
        }

        //   zk proof
        for (uint256 i = 0; i < BeforeAccountTreeRoot.length; i++) {
            uint256[4] memory input = [
                    BeforeAccountTreeRoot[i],
                    AfterAccountTreeRoot[i],
                    BeforeCEXAssetsCommitment[i],
                    AfterCEXAssetsCommitment[i]
                ];
            bool rst = verifyProof(
                a[i],
                b[i],
                c[i],
                input
            );
            require(rst,"zk proof fail");
        }

        IMainTreasury(mainTreasury).updateZKP(zkpId, AfterAccountTreeRoot[AfterAccountTreeRoot.length -
        return true;
    }
```

Recommendation

Adding access control for the `submit` function.

Alleviation Fixed

The development team fixed this issue in commit https://github.com/MUFEX-Exchange/smart-contract/commit/123e80f8f0d84d4583be57d320d0278e04c0f99b

3. Gas limitation for the `receive` function

Low risk — Security Analyzer

There is a gas limit of 2300 if the call transfer ETH to the `DepositWallet` contract by `transfer` function or `send` function.

```
    receive() external payable {
        TransferHelper.safeTransferETH(treasury, msg.value);//@audit gas fee  ?
        emit EtherCollected(treasury, msg.value, "");
    }
```

File(s) Affected

contracts/DepositWallet.sol #14-14

```
14      receive() external payable {
```

Recommendation

Adding another function to transfer ETH.

Alleviation Acknowledged

The development team responded that the receive function is only used by the EOA users.

4. DoS attack when creating a wallet

Low risk — Security Analyzer

The `DepositWalletFactory` contract creates a wallet contract for users with the salt. As a result, a malicious user can create a wallet contract if he/she knew the rule of salt before MUFEX does or front-run the transaction that MUFEX intends to execute.

File(s) Affected

contracts/DepositWalletFactory.sol #27-27

```
27          wallet = address(new DepositWallet{salt: salt}());
```

Recommendation

Checking if the factory of the create wallet contract is the right one.

Alleviation Acknowledged

The development team acknowledged this issue.

Informational (3)

1. Gas savings

Informational — Security Analyzer

For the version of commit `056df89e788c8e35f03c7a37df3eefbe81ca4127`, on May 30.

Reading a storage-type variable cost more gas than reading a memory variable.

Example A:
`solidity //DepositWalletFactory.sol function batchCreateWallets(bytes32[] memory salts, address[] memory accounts) external override returns (address[] memory wallets) { ... for (uint256 i = 0; i < salts.length; i++) { ... DepositWallet(payable(wallets[i])).initialize(accounts[i], treasury); ... } ... }` For the above example, it is gas-saving by declaring a new memory type variable `_treasury` that is assigned with `treasury`, then using the `_treasury` instead of `treasury` to save gas.

Example B:
`solidity //MainTreasury.sol function setVerifier(address verifier_) external override onlyOwner { require(verifier ==`

`address(0), "verifier already set"); verifier = verifier_; emit VerifierSet(verifier); }`

For the above example, we can use the variable `verifier_` instead of `verifier` to save gas when emitting the event.

File(s) Affected

contracts/DepositWalletFactory.sol #33-43

```
    function batchCreateWallets(bytes32[] memory salts, address[] memory accounts) external override ret
        require(salts.length == accounts.length, "length not the same");
        wallets = new address[](salts.length);
        for (uint256 i = 0; i < salts.length; i++) {
            require(getWallet[salts[i]] == address(0), "used salt");
            wallets[i] = address(new DepositWallet{salt: salts[i]}());
            DepositWallet(payable(wallets[i])).initialize(accounts[i], treasury);//@audit gas saving
            getWallet[salts[i]] = wallets[i];
        }
        emit BatchWalletsCreated(salts, accounts, wallets);
    }
```

Recommendation

Replacing the reading storage variable with the reading memory variable to save gas.

Alleviation Fixed

The development team resolved this issue in the commit https://github.com/MUFEX-Exchange/smart-contract/commit/c1300117f7696c9dc6df1363c742f56b3d623624

2. Unclear error in require logic

Informational Security Analyzer

In the given smart contract code, there are two `require` statements that use a counter variable `i` within the error messages. These error messages are not informative, and since Solidity does not have support for string interpolation, the value of `i` will not be parsed and displayed in the error message. This can cause confusion and make it difficult for developers or users to understand the actual issue.

File(s) Affected

contracts/Verifier.sol #124-127

```
        for (uint256 i = 1; i < BeforeAccountTreeRoot.length; i++) {
            require(BeforeAccountTreeRoot[i] == AfterAccountTreeRoot[i-1],"BeforeAccountTreeRoot[i] !=
            require(BeforeCEXAssetsCommitment[i] == AfterCEXAssetsCommitment[i-1],"BeforeCEXAssetsComm
        }
```

Recommendation

To address this issue, the error messages can be made more descriptive and provide some general insight into the nature of the error, without the need for parsing the counter variable. This will provide clearer information regarding the error that occurred.

Alleviation Acknowledged

The development team acknowledged this issue.

3. Potential Repeated Item Inserted into `allGeneralWithdrawnIndex` or `allForceWithdrawnIndex`

Informational Security Analyzer

For the version of commit `056df89e788c8e35f03c7a37df3eefbe81ca4127`, on May 30.

In the `MainTreasury` contract, the `isWithdrawn` checks if there is an `index` is processed or not, and the `_setWithdrawn` function marks an `index` as processed.

However, the **`allGeneralWithdrawnIndex`** array and the **`allForceWithdrawnIndex`** array may exist duplicated items since those two functions are unable to keep items of the **`allGeneralWithdrawnIndex`** array and the **`allForceWithdrawnIndex`** array to be unique.

Here is the PoC:
```` ``` ````solidity contract MainTreasuryTest is Test {

```
mapping(uint256 => uint256) private generalWithdrawnBitMap;
mapping(uint256 => uint256) private forceWithdrawnBitMap;
uint256[] private allGeneralWithdrawnIndex;
uint256[] private allForceWithdrawnIndex;

function testItemDuplicated() public {
    generalWithdraw(4609);
    generalWithdraw(4612);
    assert(allForceWithdrawnIndex.length == 2);
    assert(allForceWithdrawnIndex[0] == allForceWithdrawnIndex[1]);
}
```

//index 10010_00_000_001, -

> *"4609 //index 10010_00_000_100, → 4612 function generalWithdraw(uint256 index) public { require(!isWithdrawn(index, false), "Drop already withdrawn"); _setWithdrawn(index, false); }"*

```
function isWithdrawn(uint256 index, bool isGeneral) public view returns (bool) {
    uint256 wordIndex = index / 256;// wordIndex = 10010,       wordIndex = 10010
    uint256 bitIndex = index % 256; // bitIndex = 0_000_001,    bitIndex = 0_000_100;
    console.logString("isWithDrawn");
    console.logUint(wordIndex);
    console.logUint(bitIndex);
    uint256 word;
    if (isGeneral) {
        word = generalWithdrawnBitMap[wordIndex];
    } else {
        word = forceWithdrawnBitMap[wordIndex];
    }
    uint256 mask = (1 << bitIndex);// mask = 00010, mask = 10000
    return word & mask == mask;//
}

function _setWithdrawn(uint256 index, bool isGeneral) internal {
    uint256 wordIndex = index / 256;
    uint256 bitIndex = index % 256;
    console.logString("_setWithdrawn");
    console.logUint(wordIndex);
    console.logUint(bitIndex);
    if (isGeneral) {
        generalWithdrawnBitMap[wordIndex] = generalWithdrawnBitMap[wordIndex] | (1 << bitIndex);//
        allGeneralWithdrawnIndex.push(wordIndex);
    } else {
        forceWithdrawnBitMap[wordIndex] = forceWithdrawnBitMap[wordIndex] | (1 << bitIndex);
        allForceWithdrawnIndex.push(wordIndex);
    }
}
```

} ```` ``` ````

File(s) Affected
````

contracts/MainTreasury.sol #144-167

```
    function isWithdrawn(uint256 index, bool isGeneral) public view returns (bool) {
        uint256 wordIndex = index / 256;
        uint256 bitIndex = index % 256;
        uint256 word;
        if (isGeneral) {
            word = generalWithdrawnBitMap[wordIndex];
        } else {
            word = forceWithdrawnBitMap[wordIndex];
        }
        uint256 mask = (1 << bitIndex);
        return word & mask == mask;
    }

    function _setWithdrawn(uint256 index, bool isGeneral) internal {
        uint256 wordIndex = index / 256;
        uint256 bitIndex = index % 256;
        if (isGeneral) {
            generalWithdrawnBitMap[wordIndex] = generalWithdrawnBitMap[wordIndex] | (1 << bitIndex);//
            allGeneralWithdrawnIndex.push(wordIndex);
        } else {
            forceWithdrawnBitMap[wordIndex] = forceWithdrawnBitMap[wordIndex] | (1 << bitIndex);
            allForceWithdrawnIndex.push(wordIndex);
        }
    }
```

Recommendation

Checking if the implementation matches the design and refactoring the code if it not.

Alleviation Acknowledged

The development team responded that it is fine to have repeated `wordIndex` in the `allGeneralWithdrawnIndex` and `allForceWithdrawnIndex`.

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 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 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.