METATRUST

Security Assessment for NodeDAO IV

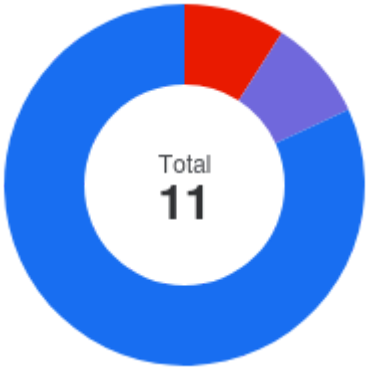
May 9, 2023

Executive Summary

Overview	
Project Name	NodeDAO IV
Codebase URL	-
Scan Engine	Security Analyzer
Scan Time	2023/05/9 16:18:20
Commit Id	-

Total	
Critical Issues	1
High risk Issues	0
Medium risk Issues	0
Low risk Issues	1
Informational Issues	9

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	9%	1
High risk Issues	0%	0
Medium risk Issues	0%	0
Low risk Issues	9%	1
Informational Issues	82%	9

Summary of Findings

MetaScan security assessment was performed on **May 9, 2023 16:18:20** on project **NodeDAO IV** with the repository on branch **default branch**. The assessment was carried out by scanning the project's codebase using the scan engine **Security Analyzer**. There are in total **11** vulnerabilities / security risks discovered during the scanning session, among which **1** critical vulnerabilities, **0** high risk vulnerabilities, **0** medium risk vulnerabilities, **1** low risk vulnerabilities, **9** informational issues.

ID	Description	Severity	Alleviation
MSA-001	Wrong Token Sent	Critical	Fixed
MSA-002	Centralized Roles	Low risk	Mitigated
MSA-003	Potential front-run risk	Informational	Acknowledged
MSA-004	Tautology	Informational	Acknowledged
MSA-005	Third-Party dependency brings potential price manipulation	Informational	Acknowledged
MSA-006	Address Input validation	Informational	Acknowledged
MSA-007	Unnecessary SafeMath usage	Informational	Acknowledged
MSA-008	Unused return value	Informational	Acknowledged
MSA-009	A require check could be by-passed	Informational	Fixed
MSA-010	Unsafe cast	Informational	Acknowledged
MSA-011	Empty functions	Informational	Acknowledged

Findings

Critical (1)

1. Wrong Token Sent

Critical | Security Analyzer

In the `HubController` contract, there is a function `withdrawPending` to withdraw tokens from the current contract. However, the tokens sent to the user are native tokens, instead of the `token`.

Wrong tokens sent to users will cause fund loss and service denial.

Meanwhile, there is no function able to receive native tokens for the `HubController`.

File(s) Affected

HubController.sol #22-34

```
22      function withdrawPending(
23          address token,
24          address user,
25          uint256 userPending
26      ) public returns (bool) {
27          require(msg.sender == vaults || msg.sender == owner(), "!vault");
28          require(address(0) != user, "user address is zero");
29          if (userPending > 0) {
30              payable(user).transfer(userPending);
31              emit Reward(token, user, userPending);
32          }
33          return true;
34      }
```

Recommendation

Recommend updating the tokens sent to users from native tokens to the `token`, or adding a function to receive native tokens.

Alleviation Fixed

The development team added a `receive` function to receive the native tokens, in the commit https://github.com/NodeDAO/KingHash-FIL/commit/ef042f17ef54421a136b057cbdba0cd4d02c309c

High risk (0)

No High risk vulnerabilities found here

Medium risk (0)

No Medium risk vulnerabilities found here

Low risk (1)

1. Centralized Roles

Low risk Security Analyzer

In the OwnableUpgradeable contract and Ownable contract, the owner has the privilege of the below functions:

- renounceOwnership: renounces ownership;
- transferOwnership: transfers ownership. Contracts that inherit the OwnableUpgradeable contract, have the same role owner:
- BeneficiaryAndVaultFactory;
- GovernanceHub;
- Vault;
- StakingBase;
- NFIL;
- LiquidStaking. The contract that inherits the Ownable contract, has the safe role owner:
- HubController. In the UUPSUpgradeable contract, the proxy has the privilege of the below functions:
- upgradeTo: upgrades the implementation;
- upgradeToAndCall: upgrades the implementation and calls to the implementation; Contracts that inherit the UUPSUpgradeable contract, have the same role proxy:
- BeneficiaryAndVaultFactory;
- GovernanceHub;
- HubPool;
- LiquidStaking;
- NFIL. In the Vault contract,
- The role governance has the privilege of the below functions:
  - setRecipient: sets the recipient;
  - setReserves: sets the reserve. In the StakingBase contract,
- The role governace has the privilege of the below function:
  - setGovernance: sets governance. In the NFIL contract,
- The role owner has the privilege of the below functions:
  - setLiquidStaking: sets liquidStakingContractAddress;
- The liquidStakingContractAddress address has the privilege of the below function:
  - whiteListMint: mints tokens to arbitrary addresses;
  - whiteListBurn: burns tokens from arbitrary addresses. In the LiquidStaking contract,
- The role governance has the privilege of the below functions:
  - addBeneficiary: adds beneficiary;
  - delBeneficiary: deletes delBeneficiary;
  - setTotalPoolFilLimit: sets totalPoolFilLimit; In the GovernanceHub contract,
- The role owner has the privilege of the below functions:
  - addAdmin: adds admin;
  - delAdmin: deletes admin;
  - transferOwner: transfers the ownership;
  - transferGovernance: transfers the governance;
  - addNodeBeneficiary: adds beneficiary for target contract;
  - delNodeBeneficiary: deletes beneficiary for target contract;
  - setTotalPoolFilLimit: sets totalPoolFilLimit for target contract;
  - setRecipient: sets recipient for target contract;
  - setReserves: sets reserve for target contract;
  - setBeneficiaryFactory: sets beneficiaryFactoryAddress;
  - setKinghashVaultContract: sets kinghashVaultContract;
  - setLiquidStakingContract: sets liquidStakingContract;
  - setOperatorAndNode: set operator and nodeid for the target contract;
  - setKingHashVault: sets vault for target contract;
  - setLiquidStaking: sets liquidStaking for target contract;
  - createBeneficiary: creates beneficiary;
  - multiCall: performances multiple calls in the multiCall function.
- The role admin has the privilege of the below functions:
  - claimRewards: claim rewards for target contract;

  - depositFil: deposits fil for the target contract;
  - withdrawFil: withdraws fil for the target contract;
  - claimBalanceRewards: claims reward;
  - withdrawBalanceFil: withdraws fil balance for the target contract;
  - setVaultPer: calls the setVaultPer function for the target contract;
  - changeBeneficiary: changes beneficiary for the target contract;
  - setLiquidStakingRewardAddress: sets LiquidStakingRewardAddress for target contract. In the contract HubController:
- The role owner has the privilege of the below functions:
  - setVault: sets vault;
  - withdrawPending: withdraws native tokens from the HubController contract to an arbitrary account.
  - inCaseTokensGetStuck: transfers any tokens to account with an arbitrary amount.
- The role vault has the privilege of the below functions:
  - withdrawPending: withdraws native tokens from the HubController contract to an arbitrary account. In the contract HubPool:
- The role governance has the privilege of the below functions:
  - setTotalAmountLimit: sets totalAmountLimit for the pool;
  - setPoolFee: sets the fee for the pool;
  - setController: sets the controller;
  - setPause: sets paused;
  - setBlockReward: sets blockReward for a pool;
  - add: adds a pool;
- The role owner has the privilege of the below functions:
  - setPause: sets paused. In the contract Beneficiary,
- The role governace has the privilege of the below functions:
  - claimRewards: claims specified rewards;
  - claimBalanceRewards: claims all the rewards;
  - depositFil: deposit fil;
  - withdrawFil: withdraws fil;
  - withdrawBalanceFil: withdraws all the fil;
  - setOperatorAndNode: sets operator and nodeId;
  - setVaultPer: sets liquidStakingPer and operatoVaultPer;
  - setKingHashVault: sets kingHashVault;
  - setOperatoVault: sets operatoVault;
  - setLiquidStaking: sets liquidStaking;
  - setLiquidStakingRewardAddress: sets liquidStakingRewardAddress;
  - changeBeneficiary: changes beneficiary; In the contract Msign,
- The role signer has the privilege of the below functions:
  - activate: activates a proposal;
  - sign: signs a proposal.

File(s) Affected

Recommendation

Consider implementing a decentralized governance mechanism or a multi-signature scheme that requires consensus among multiple parties before pausing or unpausing the contract. This can help mitigate the centralization risk associated with a single owner controlling critical contract functions. Alternatively, you can provide a clear justification for the centralization aspect and ensure that users are aware of the potential risks associated with a single point of control.

Alleviation Mitigated

The development acknowledged this issue.

Informational (9)

1. Potential front-run risk

Informational Security Analyzer

There is a potential front-run risk on the `initialize` function.

File(s) Affected

HubPool.sol #50-54

```
50      function initialize() public initializer {
51          __Ownable_init();
52          __UUPSUpgradeable_init();
53          paused = false;
54      }
```

Recommendation

Recommend checking the execution result of the initialize after deploying the contract.

Alleviation Acknowledged

The development team responded that they will check the status of the contract after the calling of the `initialize` function.

2. Tautology

Informational Security Analyzer

_amount is a uint256, so `_amount >= 0` will always be true.

There is an outer if branch to ensure that the _amount is greater than zero, so, there is no need to repeatedly recheck it again in the inner `if` branch.

File(s) Affected

HubPool.sol #222-234

```
222          if (_amount > 0) {
223              uint256 beforeToken = pool.token.balanceOf(address(this));
224              pool.token.safeTransferFrom(msg.sender, address(this), _amount);
225              uint256 afterToken = pool.token.balanceOf(address(this));
226              uint transfAmount = afterToken.sub(beforeToken);
227              require(transfAmount == _amount, "transfer amount do not eq deposit amount");
228
229              if (_amount > 0) {
230                  user.amount = user.amount.add(_amount);
231                  pool.totalAmount = pool.totalAmount.add(_amount);
232              }
233              user.lastDepostBlock = block.number;
234          }
```

Recommendation

Recommend fixing the incorrect comparison by changing the value type or the comparison.

Alleviation Acknowledged

The development team acknowledged this issue.

3. Third-Party dependency brings potential price manipulation

Informational Security Analyzer

The NodeAPI.withdrawBalance() function in the Beneficiary.sol file is marked with an audit comment that suggests it has been manipulated. A previous attack by Merlin Lab is referenced in the comment, which highlights the potential risk of manipulation. If afterBal is manipulated, it could cause an incorrect value for _amount, leading to potential losses.

File(s) Affected

Beneficiary.sol #186-192

```
186     function withdrawFil(uint256 amount, bytes calldata params) external {
187         require(amount > 0, "amount must > 0");
188         uint256 beforeBal = address(this).balance;
189         // call node api by sp account id
190         NodeAPI.withdrawBalance(uint64(nodeId), params);
191
192         uint256 afterBal = address(this).balance;
```

Recommendation

To mitigate the risk of manipulation, it is recommended to carefully review the implementation of NodeAPI.withdrawBalance() and ensure that it is secure and properly audited. Additionally, it is recommended to implement appropriate security measures to detect and prevent manipulation of afterBal, such as using an oracle or multiple sources of balance data.

Alleviation Acknowledged

WithdrawBalance is encapsulated by the node interface, and generally there will be no exceptions This function can only be called by operators Verify that the function is executed successfully by verifying the amount The return value of the function is compressed by the cbor algorithm and the verification cost is high

4. Address Input validation

Informational Security Analyzer

The `transferOwner` and `transferGovernance` functions in the smart contract code lack zero address validation. This means that the contract allows the transfer of ownership or governance to a zero address, which can result in the loss of control over the contract.

File(s) Affected

GovernanceHub.sol #71-77

```
71      function transferOwner(address target, address newOwner) external onlyOwner {
72          IGovernanceHub(target).transferOwnership(newOwner);
73      }
74
75      function transferGovernance(address target, address addr) external onlyOwner {
76          IGovernanceHub(target).setGovernance(addr);
77      }
```

Recommendation

To mitigate this issue, it is recommended to add zero address validation to the `transferOwner` and `transferGovernance` functions.

Alleviation Acknowledged

The development team acknowledged this issue.

5. Unnecessary SafeMath usage

Informational Security Analyzer

The usage of SafeMath library in the function `_claimRewards()` of `Beneficiary.sol` is unnecessary as the Solidity version used is 0.8.17, which has built-in arithmetic overflow and underflow protection.

File(s) Affected

Beneficiary.sol #131-162

```
131     }
132
133     function claimBalanceRewards(bool needReward) external {
134         uint256 amount = address(this).balance;
135         _claimRewards(needReward, amount);
136     }
137
138     function _claimRewards(bool needReward, uint256 amount) private onlyGovernance {
139         require(amount > 0, "amount must > 0");
140         // sp reward
141         uint256 _totalOperatoVaultReward = amount.mul(operatoVaultPer).div(tenThousand);
142
143         uint256 _plantReward = amount.sub(_totalOperatoVaultReward);
144         // LiquidStaking reward
145         uint256 _totalLiquidStakingReward = _plantReward.mul(liquidStakingPer).div(tenThousand);
146         // dao reward
147         uint256 _totalKingHashVaultReward = _plantReward.sub(_totalLiquidStakingReward);
148
149         if (_totalKingHashVaultReward > 0) {
150             require(kingHashVault != address(0), "kingHashVault is zero");
151             totalKingHashVaultReward = totalKingHashVaultReward.add(_totalKingHashVaultReward);
152             ILiquidStaking(kingHashVault).claimRewards{value: _totalKingHashVaultReward}(false);
153         }
154
155         if (_totalOperatoVaultReward > 0) {
156             require(operatoVault != address(0), "operatoVault is zero");
157             totalOperatoVaultReward = totalOperatoVaultReward.add(_totalOperatoVaultReward);
158             ILiquidStaking(operatoVault).claimRewards{value: _totalOperatoVaultReward}(needReward);
159         }
160
161         if (_totalLiquidStakingReward > 0) {
162             require(liquidStakingRewardAddress != address(0), "liquidStakingRewardAddress is zero");
```

Recommendation

Since Solidity 0.8.x has built-in arithmetic overflow and underflow protection, it is not necessary to use the SafeMath library. All the calls to the SafeMath functions in _claimRewards() function can be removed, and regular arithmetic operations can be used instead.

Alleviation Acknowledged

The development team acknowledged this issue.

6. Unused return value

Informational Security Analyzer

The functions `claimRewards()` and `withdrawFil()` in the `Beneficiary.sol` file both call the `NodeAPI.withdrawBalance()` function, but they do not use the return value of the function. This means that the return value is not being utilized, which could lead to unexpected results.

File(s) Affected

Beneficiary.sol #180-189

```
180
181     /**
182      * @notice withdraw sp node available balance for unstaking
183      * @param amount unstaking amount
184      * @param params withdrawBalance params cbor serialize data
185      */
186     function withdrawFil(uint256 amount, bytes calldata params) external {
187         require(amount > 0, "amount must > 0");
188         uint256 beforeBal = address(this).balance;
189         // call node api by sp account id
```

Recommendation

To avoid potential issues, it is recommended to utilize the return value of the `NodeAPI.withdrawBalance()` function in both `claimRewards()` and `withdrawFil()` functions. This can be achieved by assigning the return value to a variable or performing any required operations on it.

Alleviation Acknowledged

The development team acknowledged this issue.

7. A require check could be by-passed

Informational Security Analyzer

The `require` statement validates the parameter `to`, however, the parameter `to` is not used in the latter statements, so the caller can pass the value of the operator to the parameter `to` and bypass the `require` check.

File(s) Affected

Beneficiary.sol #175-179

```
175     function depositFil(uint256 to, uint256 amount) external onlyGovernance {
176         require(to == operator && operator > 0, "address to not allow");
177         ILiquidStaking(liquidStaking).depositFil(amount);
178         totalStakingFil = totalStakingFil.add(amount);
179     }
```

Recommendation

Checking the condition to be checked in the `require` statement.

Alleviation Fixed

The development team replied that this check is used to prevent the potential wrong call from the front end by mistake.

8. Unsafe cast

Informational Security Analyzer

The function `withdrawFil()` in the `Beneficiary.sol` file contains a potential arithmetic overflow issue, as noted by the audit comment. The nodeId variable is of type uint256, but it is being cast to uint64 when calling the `NodeAPI.withdrawBalance()` function. This type cast could cause an overflow if nodeId exceeds the maximum value of uint64.

File(s) Affected

Beneficiary.sol #1-1

```
1  // SPDX-License-Identifier: UNLICENSED
```

Recommendation

To avoid potential overflow issues, it is recommended to update the `NodeAPI.withdrawBalance()` function to accept a `uint256` value for the nodeId parameter instead of a `uint64` value. Alternatively, the nodeId variable could be checked to ensure that it does not exceed the maximum value of `uint64` before being cast.

Alleviation Acknowledged

The development team acknowledged this issue.

9. Empty functions

Informational Security Analyzer

In the HubController contract, there are two empty functions, which are intended to be invoked in the HubPool contract:

```
function withdrawWithPid(uint256 _pid, uint256 _amount) public notPause {
    ...
    IController(controller).withdraw(address(pool.token), _amount.sub(poolBalance));
    ...
}

function earn(address token) public {
    approveCtr(token);
    IController(controller).earn(token);
}
```

It makes no sense to invoke the two empty functions, `withdraw` and `earn`.

File(s) Affected

HubController.sol #18-20

```
18      function withdraw(address _token, uint _amount) public {}
19
20      function earn(address _token) public {}
```

Recommendation

Recommend adding implementations for the `withdraw` function and the `earn` function.

Alleviation Acknowledged

The development team acknowledged this issue.

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.