METATRUST

Draft
Security Assessment for

52-LogicBug-Vader2 (StaticFail) (3OK-SP)

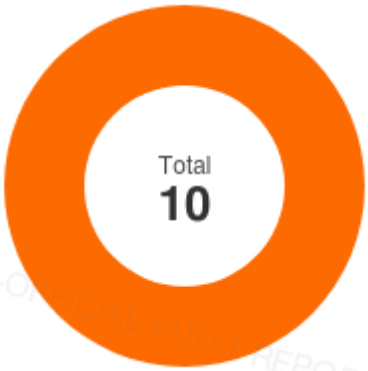
July 23, 2023

Executive Summary

Overview	
Project Name	52-LogicBug-Vader2 (StaticFail) (3OK-SP)
Codebase URL	https://github.com/metatrust-demo/LogicBug-Vader2
Scan Engine	AI Analyzer
Scan Time	2023/07/23 22:33:36
Commit Id	9913cbf

Total	
Critical Issues	0
High risk Issues	10
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%	10
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 22:33:36** on project **52-LogicBug-Vader2 (StaticFail) (3OK-SP)** with the repository **https://github.com/metatrust-demo/LogicBug-Vader2** 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 **10** vulnerabilities / security risks discovered during the scanning session, among which **0** critical vulnerabilities, **10** 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-200: Insecure LP Token Value Calculation	High risk
MSA-005	MWE-206: No Slippage Limit Check	High risk
MSA-006	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-007	MWE-206: No Slippage Limit Check	High risk
MSA-008	MWE-206: No Slippage Limit Check	High risk
MSA-009	MWE-206: No Slippage Limit Check	High risk
MSA-010	MWE-206: No Slippage Limit Check	High risk

Findings

Critical (0)

No Critical vulnerabilities found here

High risk (10)

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/dex-v2/pool/VaderPoolV2.sol #126-167

```
126     function mintSynth(
127         IERC20 foreignAsset,
128         uint256 nativeDeposit,
129         address from,
130         address to
131     )
132         external
133         override
134         nonReentrant
135         supportedToken(foreignAsset)
136         returns (uint256 amountSynth)
137     {
138         nativeAsset.safeTransferFrom(from, address(this), nativeDeposit);
139
140         ISynth synth = synthFactory.synths(foreignAsset);
141
142         if (synth == ISynth(_ZERO_ADDRESS))
143             synth = synthFactory.createSynth(
144                 IERC20Extended(address(foreignAsset))
145             );
146
147         (uint112 reserveNative, uint112 reserveForeign, ) = getReserves(
148             foreignAsset
149         ); // gas savings
150
151         amountSynth = VaderMath.calculateSwap(
152             nativeDeposit,
153             reserveNative,
154             reserveForeign
155         );
156
157         // TODO: Clarify
158         _update(
159             foreignAsset,
160             reserveNative + nativeDeposit,
161             reserveForeign,
162             reserveNative,
163             reserveForeign
164         );
165
166         synth.mint(to, amountSynth);
167     }
```

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/dex-v2/pool/VaderPoolV2.sol #179-219

```
179     function burnSynth(
180         IERC20 foreignAsset,
181         uint256 synthAmount,
182         address to
183     ) external override nonReentrant returns (uint256 amountNative) {
184         ISynth synth = synthFactory.synths(foreignAsset);
185
186         require(
187             synth != ISynth(_ZERO_ADDRESS),
188             "VaderPoolV2::burnSynth: Inexistent Synth"
189         );
190
191         require(
192             synthAmount > 0,
193             "VaderPoolV2::burnSynth: Insufficient Synth Amount"
194         );
195
196         IERC20(synth).safeTransferFrom(msg.sender, address(this), synthAmount);
197         synth.burn(synthAmount);
198
199         (uint112 reserveNative, uint112 reserveForeign, ) = getReserves(
200             foreignAsset
201         ); // gas savings
202
203         amountNative = VaderMath.calculateSwap(
204             synthAmount,
205             reserveForeign,
206             reserveNative
207         );
208
209         // TODO: Clarify
210         _update(
211             foreignAsset,
212             reserveNative - amountNative,
213             reserveForeign,
214             reserveNative,
215             reserveForeign
216         );
217
218         nativeAsset.safeTransfer(to, amountNative);
219     }
```

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/dex-v2/pool/VaderPoolV2.sol #284-335

```
    function mintFungible(
        IERC20 foreignAsset,
        uint256 nativeDeposit,
        uint256 foreignDeposit,
        address from,
        address to
    ) external override nonReentrant returns (uint256 liquidity) {
        IERC20Extended lp = wrapper.tokens(foreignAsset);

        require(
            lp != IERC20Extended(_ZERO_ADDRESS),
            "VaderPoolV2::mintFungible: Unsupported Token"
        );

        (uint112 reserveNative, uint112 reserveForeign, ) = getReserves(
            foreignAsset
        ); // gas savings

        nativeAsset.safeTransferFrom(from, address(this), nativeDeposit);
        foreignAsset.safeTransferFrom(from, address(this), foreignDeposit);

        PairInfo storage pair = pairInfo[foreignAsset];
        uint256 totalLiquidityUnits = pair.totalSupply;
        if (totalLiquidityUnits == 0) liquidity = nativeDeposit;
        else
            liquidity = VaderMath.calculateLiquidityUnits(
                nativeDeposit,
                reserveNative,
                foreignDeposit,
                reserveForeign,
                totalLiquidityUnits
            );

        require(
            liquidity > 0,
            "VaderPoolV2::mintFungible: Insufficient Liquidity Provided"
        );

        pair.totalSupply = totalLiquidityUnits + liquidity;

        _update(
            foreignAsset,
            reserveNative + nativeDeposit,
            reserveForeign + foreignDeposit,
            reserveNative,
            reserveForeign
        );

        lp.mint(to, liquidity);

        emit Mint(from, to, nativeDeposit, foreignDeposit);
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

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/dex-v2/pool/BasePoolV2.sol #250-293

```
250     function _burn(uint256 id, address to)
251         internal
252         nonReentrant
253         returns (uint256 amountNative, uint256 amountForeign)
254     {
255         require(
256             ownerOf(id) == address(this),
257             "BasePoolV2::burn: Incorrect Ownership"
258         );
259
260         IERC20 foreignAsset = positions[id].foreignAsset;
261
262         (uint112 reserveNative, uint112 reserveForeign, ) = getReserves(
263             foreignAsset
264         ); // gas savings
265
266         uint256 liquidity = positions[id].liquidity;
267
268         PairInfo storage pair = pairInfo[foreignAsset];
269         uint256 _totalSupply = pair.totalSupply;
270         amountNative = (liquidity * reserveNative) / _totalSupply;
271         amountForeign = (liquidity * reserveForeign) / _totalSupply;
272
273         require(
274             amountNative > 0 && amountForeign > 0,
275             "BasePoolV2::burn: Insufficient Liquidity Burned"
276         );
277
278         pair.totalSupply = _totalSupply - liquidity;
279         _burn(id);
280
281         nativeAsset.safeTransfer(to, amountNative);
282         foreignAsset.safeTransfer(to, amountForeign);
283
284         _update(
285             foreignAsset,
286             reserveNative - amountNative,
287             reserveForeign - amountForeign,
288             reserveNative,
289             reserveForeign
290         );
291
292         emit Burn(msg.sender, amountNative, amountForeign, to);
293     }
```

Recommendation

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

5. MWE-206: No Slippage Limit Check

High risk Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

contracts/dex-v2/router/VaderRouterV2.sol #114-151

```
114     function addLiquidity(
115         IERC20 tokenA,
116         IERC20 tokenB,
117         uint256 amountADesired,
118         uint256 amountBDesired,
119         address to,
120         uint256 deadline
121     ) public override ensure(deadline) returns (uint256 liquidity) {
122         IERC20 foreignAsset;
123         uint256 nativeDeposit;
124         uint256 foreignDeposit;
125 
126         if (tokenA == nativeAsset) {
127             require(
128                 pool.supported(tokenB),
129                 "VaderRouterV2::addLiquidity: Unsupported Assets Specified"
130             );
131             foreignAsset = tokenB;
132             foreignDeposit = amountBDesired;
133             nativeDeposit = amountADesired;
134         } else {
135             require(
136                 tokenB == nativeAsset && pool.supported(tokenA),
137                 "VaderRouterV2::addLiquidity: Unsupported Assets Specified"
138             );
139             foreignAsset = tokenA;
140             foreignDeposit = amountADesired;
141             nativeDeposit = amountBDesired;
142         }
143 
144         liquidity = pool.mint(
145             foreignAsset,
146             nativeDeposit,
147             foreignDeposit,
148             msg.sender,
149             to
150         );
151     }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

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/dex/pool/BasePool.sol #214-253

```
214     function _burn(uint256 id, address to)
215         internal
216         nonReentrant
217         returns (uint256 amountNative, uint256 amountForeign)
218     {
219         require(
220             ownerOf(id) == address(this),
221             "BasePool::burn: Incorrect Ownership"
222         );
223
224         (uint112 reserveNative, uint112 reserveForeign, ) = getReserves(); // gas savings
225         IERC20 _nativeAsset = nativeAsset; // gas savings
226         IERC20 _foreignAsset = foreignAsset; // gas savings
227         uint256 nativeBalance = IERC20(_nativeAsset).balanceOf(address(this));
228         uint256 foreignBalance = IERC20(_foreignAsset).balanceOf(address(this));
229
230         uint256 liquidity = positions[id].liquidity;
231
232         uint256 _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can
233         amountNative = (liquidity * nativeBalance) / _totalSupply; // using balances ensures pro-rata
234         amountForeign = (liquidity * foreignBalance) / _totalSupply; // using balances ensures pro-rata
235
236         require(
237             amountNative > 0 && amountForeign > 0,
238             "BasePool::burn: Insufficient Liquidity Burned"
239         );
240
241         totalSupply -= liquidity;
242         _burn(id);
243
244         _nativeAsset.safeTransfer(to, amountNative);
245         _foreignAsset.safeTransfer(to, amountForeign);
246
247         nativeBalance = _nativeAsset.balanceOf(address(this));
248         foreignBalance = _foreignAsset.balanceOf(address(this));
249
250         _update(nativeBalance, foreignBalance, reserveNative, reserveForeign);
251
252         emit Burn(msg.sender, amountNative, amountForeign, to);
253     }
```

contracts/dex-v2/pool/BasePoolV2.sol #250-293

```
    function _burn(uint256 id, address to)
        internal
        nonReentrant
        returns (uint256 amountNative, uint256 amountForeign)
    {
        require(
            ownerOf(id) == address(this),
            "BasePoolV2::burn: Incorrect Ownership"
        );

        IERC20 foreignAsset = positions[id].foreignAsset;

        (uint112 reserveNative, uint112 reserveForeign, ) = getReserves(
            foreignAsset
        ); // gas savings

        uint256 liquidity = positions[id].liquidity;

        PairInfo storage pair = pairInfo[foreignAsset];
        uint256 _totalSupply = pair.totalSupply;
        amountNative = (liquidity * reserveNative) / _totalSupply;
        amountForeign = (liquidity * reserveForeign) / _totalSupply;

        require(
            amountNative > 0 && amountForeign > 0,
            "BasePoolV2::burn: Insufficient Liquidity Burned"
        );

        pair.totalSupply = _totalSupply - liquidity;
        _burn(id);

        nativeAsset.safeTransfer(to, amountNative);
        foreignAsset.safeTransfer(to, amountForeign);

        _update(
            foreignAsset,
            reserveNative - amountNative,
            reserveForeign - amountForeign,
            reserveNative,
            reserveForeign
        );

        emit Burn(msg.sender, amountNative, amountForeign, to);
    }
```

Recommendation

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

7. MWE-206: No Slippage Limit Check

High risk Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

contracts/dex/math/VaderMath.sol #117-150

```
117     function calculateSwapReverse(
118         uint256 amountOut,
119         uint256 reserveIn,
120         uint256 reserveOut
121     ) public pure returns (uint256 amountIn) {
122         // X * Y
123         uint256 XY = reserveIn * reserveOut;
124
125         // 2y
126         uint256 y2 = amountOut * 2;
127
128         // 4y
129         uint256 y4 = y2 * 2;
130
131         require(
132             y4 < reserveOut,
133             "VaderMath::calculateSwapReverse: Desired Output Exceeds Maximum Output Possible (1/4 of L
134         );
135
136         // root(-X^2 * Y * (4y - Y))     =>     root(X^2 * Y * (Y - 4y)) as Y - 4y >= 0     =>     Y >= 4y
137         uint256 numeratorA = root(XY) * root(reserveIn * (reserveOut - y4));
138
139         // X * (2y - Y)     =>     2yX - XY
140         uint256 numeratorB = y2 * reserveIn;
141         uint256 numeratorC = XY;
142
143         // -1 * (root(-X^2 * Y * (4y - Y)) + (X * (2y - Y)))     =>     -1 * (root(X^2 * Y * (Y - 4y)) +
144         uint256 numerator = numeratorC - numeratorA - numeratorB;
145
146         // 2y
147         uint256 denominator = y2;
148
149         amountIn = numerator / denominator;
150     }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

8. MWE-206: No Slippage Limit Check

High risk Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

contracts/dex/router/VaderRouter.sol #123-150

```
123     function addLiquidity(
124         IERC20 tokenA,
125         IERC20 tokenB,
126         uint256 amountADesired,
127         uint256 amountBDesired,
128         address to,
129         uint256 deadline
130     )
131         public
132         override
133         ensure(deadline)
134         returns (
135             uint256 amountA,
136             uint256 amountB,
137             uint256 liquidity
138         )
139     {
140         IVaderPool pool;
141         (pool, amountA, amountB) = _addLiquidity(
142             address(tokenA),
143             address(tokenB),
144             amountADesired,
145             amountBDesired
146         );
147         tokenA.safeTransferFrom(msg.sender, address(pool), amountA);
148         tokenB.safeTransferFrom(msg.sender, address(pool), amountB);
149         liquidity = pool.mint(to);
150     }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

9. MWE-206: No Slippage Limit Check

High risk Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

contracts/dex/router/VaderRouter.sol #394-438

```
    function calculateInGivenOut(uint256 amountOut, address[] calldata path)
        public
        view
        returns (uint256 amountIn)
    {
        if (path.length == 2) {
            address nativeAsset = factory.nativeAsset();
            IVaderPool pool = factory.getPool(path[0], path[1]);
            (uint256 nativeReserve, uint256 foreignReserve, ) = pool
                .getReserves();
            if (path[0] == nativeAsset) {
                return
                    VaderMath.calculateSwapReverse(
                        amountOut,
                        nativeReserve,
                        foreignReserve
                    );
            } else {
                return
                    VaderMath.calculateSwapReverse(
                        amountOut,
                        foreignReserve,
                        nativeReserve
                    );
            }
        } else {
            IVaderPool pool0 = factory.getPool(path[0], path[1]);
            IVaderPool pool1 = factory.getPool(path[1], path[2]);
            (uint256 nativeReserve0, uint256 foreignReserve0, ) = pool0
                .getReserves();
            (uint256 nativeReserve1, uint256 foreignReserve1, ) = pool1
                .getReserves();

            return
                VaderMath.calculateSwapReverse(
                    VaderMath.calculateSwapReverse(
                        amountOut,
                        nativeReserve1,
                        foreignReserve1
                    ),
                    foreignReserve0,
                    nativeReserve0
                );
        }
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

10. MWE-206: No Slippage Limit Check

High risk Security Analyzer

No slippage limit check was performed to prevent sandwich attacks.

File(s) Affected

contracts/dex/router/VaderRouter.sol #453-497

```
    function calculateOutGivenIn(uint256 amountIn, address[] calldata path)
        external
        view
        returns (uint256 amountOut)
    {
        if (path.length == 2) {
            address nativeAsset = factory.nativeAsset();
            IVaderPool pool = factory.getPool(path[0], path[1]);
            (uint256 nativeReserve, uint256 foreignReserve, ) = pool
                .getReserves();
            if (path[0] == nativeAsset) {
                return
                    VaderMath.calculateSwap(
                        amountIn,
                        nativeReserve,
                        foreignReserve
                    );
            } else {
                return
                    VaderMath.calculateSwap(
                        amountIn,
                        foreignReserve,
                        nativeReserve
                    );
            }
        } else {
            IVaderPool pool0 = factory.getPool(path[0], path[1]);
            IVaderPool pool1 = factory.getPool(path[1], path[2]);
            (uint256 nativeReserve0, uint256 foreignReserve0, ) = pool0
                .getReserves();
            (uint256 nativeReserve1, uint256 foreignReserve1, ) = pool1
                .getReserves();

            return
                VaderMath.calculateSwap(
                    VaderMath.calculateSwap(
                        amountIn,
                        nativeReserve1,
                        foreignReserve1
                    ),
                    foreignReserve0,
                    nativeReserve0
                );
        }
    }
```

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 52-LogicBug-Vader2 (StaticFail) (3OK-SP) 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 52-LogicBug-Vader2 (StaticFail) (3OK-SP) 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.