METATRUST

Draft

Security Assessment for

70-2021-12-vader (3New-FD) (XOK-SP) (OK-FLP) (1Positive-UT) (1Negative-FR)

July 23, 2023

Executive Summary

Overview	
Project Name	70-2021-12-vader (3New-FD) (XOK-SP) (OK-FLP) (1Positive-UT) (1Negative-FR)
Codebase URL	https://github.com/code-423n4/2021-12-vader
Scan Engine	AI Analyzer
Scan Time	2023/07/23 17:09:15
Commit Id	842662a

Total	
Critical Issues	0
High risk Issues	22
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%	22
Medium risk Issues	0%	0
Low risk Issues	0%	0
Informational Issues	0%	0

Summary of Findings

MetaScan security assessment was performed on **July 23, 2023 17:09:15** on project **70-2021-12-vader (3New-FD) (XOK-SP) (OK-FLP) (1Positive-UT) (1Negative-FR)** with the repository **https://github.com/code-423n4/2021-12-vader** 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 **22** vulnerabilities / security risks discovered during the scanning session, among which **0** critical vulnerabilities, **22** high risk vulnerabilities, **0** medium risk vulnerabilities, **0** low risk vulnerabilities, **0** informational issues.

ID	Description	Severity
MSA-001	MWE-204: Unsafe First Deposit	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
MSA-005	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-006	MWE-206: No Slippage Limit Check	High risk
MSA-007	MWE-206: No Slippage Limit Check	High risk
MSA-008	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-009	MWE-206: No Slippage Limit Check	High risk
MSA-010	MWE-207: Unauthorized Transfer	High risk
MSA-011	MWE-206: No Slippage Limit Check	High risk
MSA-012	MWE-205: Front Running	High risk
MSA-013	MWE-206: No Slippage Limit Check	High risk
MSA-014	MWE-200: Insecure LP Token Value Calculation	High risk
MSA-015	MWE-206: No Slippage Limit Check	High risk
MSA-016	MWE-206: No Slippage Limit Check	High risk
MSA-017	MWE-206: No Slippage Limit Check	High risk
MSA-018	MWE-206: No Slippage Limit Check	High risk
MSA-019	MWE-206: No Slippage Limit Check	High risk
MSA-020	MWE-206: No Slippage Limit Check	High risk
MSA-021	MWE-200: Insecure LP Token Value Calculation	High risk

ID	Description	Severity
MSA-022	MWE-200: Insecure LP Token Value Calculation	High risk

Findings

Critical (0)

No Critical vulnerabilities found here

High risk (22)

1. MWE-204: Unsafe First Deposit

High risk Security Analyzer

First depositor can break minting of shares or drain the liquidity of all users.

File(s) Affected

contracts/x-vader/XVader.sol #25-41

```
function enter(uint256 _amount) external {
    // Gets the amount of vader locked in the contract
    uint256 totalVader = vader.balanceOf(address(this));
    // Gets the amount of xVader in existence
    uint256 totalShares = totalSupply();

    uint256 xVADERToMint = totalShares == 0 || totalVader == 0 // If no xVader exists, mint it 1:1 t
        ? _amount // Calculate and mint the amount of xVader the vader is worth.
        : // The ratio will change overtime, as xVader is burned/minted and
        // vader deposited + gained from fees / withdrawn.
        (_amount * totalShares) / totalVader;

    _mint(msg.sender, xVADERToMint);

    // Lock the vader in the contract
    vader.transferFrom(msg.sender, address(this), _amount);
}
```

Recommendation

When totalSupply() == 0, send the first min liquidity LP tokens to the zero address to enable share dilution.

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 #153-194

```
    function mintSynth(
        IERC20 foreignAsset,
        uint256 nativeDeposit,
        address from,
        address to
    )
        external
        override
        nonReentrant
        supportedToken(foreignAsset)
        returns (uint256 amountSynth)
    {
        nativeAsset.safeTransferFrom(from, address(this), nativeDeposit);

        ISynth synth = synthFactory.synths(foreignAsset);

        if (synth == ISynth(_ZERO_ADDRESS))
            synth = synthFactory.createSynth(
                IERC20Extended(address(foreignAsset))
            );

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

        amountSynth = VaderMath.calculateSwap(
            nativeDeposit,
            reserveNative,
            reserveForeign
        );

        // TODO: Clarify
        _update(
            foreignAsset,
            reserveNative + nativeDeposit,
            reserveForeign,
            reserveNative,
            reserveForeign
        );

        synth.mint(to, amountSynth);
    }
```

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 #206-246

```
    function burnSynth(
        IERC20 foreignAsset,
        uint256 synthAmount,
        address to
    ) external override nonReentrant returns (uint256 amountNative) {
        ISynth synth = synthFactory.synths(foreignAsset);

        require(
            synth != ISynth(_ZERO_ADDRESS),
            "VaderPoolV2::burnSynth: Inexistent Synth"
        );

        require(
            synthAmount > 0,
            "VaderPoolV2::burnSynth: Insufficient Synth Amount"
        );

        IERC20(synth).safeTransferFrom(msg.sender, address(this), synthAmount);
        synth.burn(synthAmount);

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

        amountNative = VaderMath.calculateSwap(
            synthAmount,
            reserveForeign,
            reserveNative
        );

        // TODO: Clarify
        _update(
            foreignAsset,
            reserveNative - amountNative,
            reserveForeign,
            reserveNative,
            reserveForeign
        );

        nativeAsset.safeTransfer(to, amountNative);
    }
```

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/dex-v2/pool/VaderPoolV2.sol #311-362

```
    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.

5. MWE-200: Insecure LP Token Value Calculation

High risk Security Analyzer

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

File(s) Affected

contracts/dex-v2/pool/BasePoolV2.sol #203-246

```
    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.

6. 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/BasePoolV2.sol #479-533

```
    function _mint(
        IERC20 foreignAsset,
        uint256 nativeDeposit,
        uint256 foreignDeposit,
        address from,
        address to
    ) internal nonReentrant returns (uint256 liquidity) {
        (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,
            "BasePoolV2::mint: Insufficient Liquidity Provided"
        );

        uint256 id = positionId++;

        pair.totalSupply = totalLiquidityUnits + liquidity;
        _mint(to, id);

        positions[id] = Position(
            foreignAsset,
            block.timestamp,
            liquidity,
            nativeDeposit,
            foreignDeposit
        );

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

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

Recommendation

Add slippage limit check when do liquidity-related operations.

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-v2/pool/BasePoolV2.sol #479-533

```
    function _mint(
        IERC20 foreignAsset,
        uint256 nativeDeposit,
        uint256 foreignDeposit,
        address from,
        address to
    ) internal nonReentrant returns (uint256 liquidity) {
        (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,
            "BasePoolV2::mint: Insufficient Liquidity Provided"
        );

        uint256 id = positionId++;

        pair.totalSupply = totalLiquidityUnits + liquidity;
        _mint(to, id);

        positions[id] = Position(
            foreignAsset,
            block.timestamp,
            liquidity,
            nativeDeposit,
            foreignDeposit
        );

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

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

contracts/dex/pool/BasePool.sol #148-193

```
    function mint(address to)
        external
        override
        nonReentrant
        returns (uint256 liquidity)
    {
        (uint112 reserveNative, uint112 reserveForeign, ) = getReserves(); // gas savings
        uint256 balanceNative = nativeAsset.balanceOf(address(this));
        uint256 balanceForeign = foreignAsset.balanceOf(address(this));
        uint256 nativeDeposit = balanceNative - reserveNative;
        uint256 foreignDeposit = balanceForeign - reserveForeign;

        uint256 totalLiquidityUnits = totalSupply;
        if (totalLiquidityUnits == 0)
            liquidity = nativeDeposit; // TODO: Contact ThorChain on proper approach
        else
            liquidity = VaderMath.calculateLiquidityUnits(
                nativeDeposit,
                reserveNative,
                foreignDeposit,
                reserveForeign,
                totalLiquidityUnits
            );

        require(
            liquidity > 0,
            "BasePool::mint: Insufficient Liquidity Provided"
        );

        uint256 id = positionId++;

        totalSupply += liquidity;
        _mint(to, id);

        positions[id] = Position(
            block.timestamp,
            liquidity,
            nativeDeposit,
            foreignDeposit
        );

        _update(balanceNative, balanceForeign, reserveNative, reserveForeign);

        emit Mint(msg.sender, to, nativeDeposit, foreignDeposit);
        emit PositionOpened(msg.sender, id, liquidity);
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

8. MWE-200: Insecure LP Token Value Calculation

High risk Security Analyzer

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

File(s) Affected

contracts/mocks/MockUniswapV2Router.sol #35-80

```
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin
    ) internal virtual returns (uint256 amountA, uint256 amountB) {
        // create the pair if it doesn't exist yet
        if (IUniswapV2Factory(factory).getPair(tokenA, tokenB) == address(0)) {
            IUniswapV2Factory(factory).createPair(tokenA, tokenB);
        }
        (uint256 reserveA, uint256 reserveB) = UniswapV2Library.getReserves(
            factory,
            tokenA,
            tokenB
        );
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint256 amountBOptimal = UniswapV2Library.quote(
                amountADesired,
                reserveA,
                reserveB
            );
            if (amountBOptimal <= amountBDesired) {
                require(
                    amountBOptimal >= amountBMin,
                    "UniswapV2Router: INSUFFICIENT_B_AMOUNT"
                );
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint256 amountAOptimal = UniswapV2Library.quote(
                    amountBDesired,
                    reserveB,
                    reserveA
                );
                assert(amountAOptimal <= amountADesired);
                require(
                    amountAOptimal >= amountAMin,
                    "UniswapV2Router: INSUFFICIENT_A_AMOUNT"
                );
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }
```

contracts/mocks/MockUniswapV2Router.sol #82-114

```
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        )
    {
        (amountA, amountB) = _addLiquidity(
            tokenA,
            tokenB,
            amountADesired,
            amountBDesired,
            amountAMin,
            amountBMin
        );
        address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IUniswapV2Pair(pair).mint(to);
    }
```

Recommendation

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

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/mocks/MockUniswapV2Router.sol #186-212

```
    function removeLiquidityETH(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    )
        public
        virtual
        override
        ensure(deadline)
        returns (uint256 amountToken, uint256 amountETH)
    {
        (amountToken, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
```

contracts/mocks/MockUniswapV2Router.sol #249-285

```
    function removeLiquidityETHWithPermit(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    )
        external
        virtual
        override
        returns (uint256 amountToken, uint256 amountETH)
    {
        address pair = UniswapV2Library.pairFor(factory, token, WETH);
        uint256 value = approveMax ? type(uint256).max : liquidity;
        IUniswapV2Pair(pair).permit(
            msg.sender,
            address(this),
            value,
            deadline,
            v,
            r,
            s
        );
        (amountToken, amountETH) = removeLiquidityETH(
            token,
            liquidity,
            amountTokenMin,
            amountETHMin,
            to,
            deadline
        );
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

10. MWE-207: Unauthorized Transfer

High risk Security Analyzer

The contract allows transferring tokens from an address different from the message sender without checking the approval of the address owner.

File(s) Affected

contracts/mocks/MockUniswapV2Router.sol #605-627

```
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable virtual override ensure(deadline) {
        require(path[0] == WETH, "UniswapV2Router: INVALID_PATH");
        uint256 amountIn = msg.value;
        IWETH(WETH).deposit{value: amountIn}();
        assert(
            IWETH(WETH).transfer(
                UniswapV2Library.pairFor(factory, path[0], path[1]),
                amountIn
            )
        );
        uint256 balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >=
                amountOutMin,
            "UniswapV2Router: INSUFFICIENT_OUTPUT_AMOUNT"
        );
    }
```

Recommendation

Check the business logic about the transfer and add the approval check if necessary.

11. 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/pool/BasePool.sol #148-193

```
    function mint(address to)
        external
        override
        nonReentrant
        returns (uint256 liquidity)
    {
        (uint112 reserveNative, uint112 reserveForeign, ) = getReserves(); // gas savings
        uint256 balanceNative = nativeAsset.balanceOf(address(this));
        uint256 balanceForeign = foreignAsset.balanceOf(address(this));
        uint256 nativeDeposit = balanceNative - reserveNative;
        uint256 foreignDeposit = balanceForeign - reserveForeign;

        uint256 totalLiquidityUnits = totalSupply;
        if (totalLiquidityUnits == 0)
            liquidity = nativeDeposit; // TODO: Contact ThorChain on proper approach
        else
            liquidity = VaderMath.calculateLiquidityUnits(
                nativeDeposit,
                reserveNative,
                foreignDeposit,
                reserveForeign,
                totalLiquidityUnits
            );

        require(
            liquidity > 0,
            "BasePool::mint: Insufficient Liquidity Provided"
        );

        uint256 id = positionId++;

        totalSupply += liquidity;
        _mint(to, id);

        positions[id] = Position(
            block.timestamp,
            liquidity,
            nativeDeposit,
            foreignDeposit
        );

        _update(balanceNative, balanceForeign, reserveNative, reserveForeign);

        emit Mint(msg.sender, to, nativeDeposit, foreignDeposit);
        emit PositionOpened(msg.sender, id, liquidity);
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

12. MWE-205: Front Running

High risk Security Analyzer

Users are required to transfer assets in advance and minting token/liquidity/earning thus could be frontrun.

File(s) Affected

contracts/dex/pool/BasePool.sol #148-193

```
    function mint(address to)
        external
        override
        nonReentrant
        returns (uint256 liquidity)
    {
        (uint112 reserveNative, uint112 reserveForeign, ) = getReserves(); // gas savings
        uint256 balanceNative = nativeAsset.balanceOf(address(this));
        uint256 balanceForeign = foreignAsset.balanceOf(address(this));
        uint256 nativeDeposit = balanceNative - reserveNative;
        uint256 foreignDeposit = balanceForeign - reserveForeign;

        uint256 totalLiquidityUnits = totalSupply;
        if (totalLiquidityUnits == 0)
            liquidity = nativeDeposit; // TODO: Contact ThorChain on proper approach
        else
            liquidity = VaderMath.calculateLiquidityUnits(
                nativeDeposit,
                reserveNative,
                foreignDeposit,
                reserveForeign,
                totalLiquidityUnits
            );

        require(
            liquidity > 0,
            "BasePool::mint: Insufficient Liquidity Provided"
        );

        uint256 id = positionId++;

        totalSupply += liquidity;
        _mint(to, id);

        positions[id] = Position(
            block.timestamp,
            liquidity,
            nativeDeposit,
            foreignDeposit
        );

        _update(balanceNative, balanceForeign, reserveNative, reserveForeign);

        emit Mint(msg.sender, to, nativeDeposit, foreignDeposit);
        emit PositionOpened(msg.sender, id, liquidity);
    }
```

Recommendation

Put asset transfering and token minting in the same function to keep atomicity.

13. 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/pool/BasePool.sol #148-193

```
    function mint(address to)
        external
        override
        nonReentrant
        returns (uint256 liquidity)
    {
        (uint112 reserveNative, uint112 reserveForeign, ) = getReserves(); // gas savings
        uint256 balanceNative = nativeAsset.balanceOf(address(this));
        uint256 balanceForeign = foreignAsset.balanceOf(address(this));
        uint256 nativeDeposit = balanceNative - reserveNative;
        uint256 foreignDeposit = balanceForeign - reserveForeign;

        uint256 totalLiquidityUnits = totalSupply;
        if (totalLiquidityUnits == 0)
            liquidity = nativeDeposit; // TODO: Contact ThorChain on proper approach
        else
            liquidity = VaderMath.calculateLiquidityUnits(
                nativeDeposit,
                reserveNative,
                foreignDeposit,
                reserveForeign,
                totalLiquidityUnits
            );

        require(
            liquidity > 0,
            "BasePool::mint: Insufficient Liquidity Provided"
        );

        uint256 id = positionId++;

        totalSupply += liquidity;
        _mint(to, id);

        positions[id] = Position(
            block.timestamp,
            liquidity,
            nativeDeposit,
            foreignDeposit
        );

        _update(balanceNative, balanceForeign, reserveNative, reserveForeign);

        emit Mint(msg.sender, to, nativeDeposit, foreignDeposit);
        emit PositionOpened(msg.sender, id, liquidity);
    }
```

contracts/mocks/MockUniswapV2Router.sol #82-114

```
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        )
     {
        (amountA, amountB) = _addLiquidity(
            tokenA,
            tokenB,
            amountADesired,
            amountBDesired,
            amountAMin,
            amountBMin
        );
        address pair = UniswapV2Library.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IUniswapV2Pair(pair).mint(to);
     }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

14. 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 #213-252

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

        (uint112 reserveNative, uint112 reserveForeign, ) = getReserves(); // gas savings
        IERC20 _nativeAsset = nativeAsset; // gas savings
        IERC20 _foreignAsset = foreignAsset; // gas savings
        uint256 nativeBalance = IERC20(_nativeAsset).balanceOf(address(this));
        uint256 foreignBalance = IERC20(_foreignAsset).balanceOf(address(this));

        uint256 liquidity = positions[id].liquidity;

        uint256 _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can
        amountNative = (liquidity * nativeBalance) / _totalSupply; // using balances ensures pro-rata d
        amountForeign = (liquidity * foreignBalance) / _totalSupply; // using balances ensures pro-rata

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

        totalSupply -= liquidity;
        _burn(id);

        _nativeAsset.safeTransfer(to, amountNative);
        _foreignAsset.safeTransfer(to, amountForeign);

        nativeBalance = _nativeAsset.balanceOf(address(this));
        foreignBalance = _foreignAsset.balanceOf(address(this));

        _update(nativeBalance, foreignBalance, reserveNative, reserveForeign);

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

contracts/tokens/USDV.sol #100-120

```
    function burn(uint256 uAmount)
        external
        onlyWhenNotLocked
        returns (uint256 vAmount)
    {
        uint256 uPrice = lbt.getUSDVPrice();

        _burn(msg.sender, uAmount);

        vAmount = (uPrice * uAmount) / 1e18;

        if (exchangeFee != 0) {
            uint256 fee = (vAmount * exchangeFee) / _MAX_BASIS_POINTS;
            vAmount = vAmount - fee;
            vader.mint(owner(), fee);
        }

        vader.mint(address(this), vAmount);

        _createLock(LockTypes.VADER, vAmount);
    }
```

Recommendation

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

15. 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/pool/BasePool.sol #288-378

```
    function swap(
        uint256 nativeAmountIn,
        uint256 foreignAmountIn,
        address to
    ) public override nonReentrant validateGas returns (uint256) {
        require(
            (nativeAmountIn > 0 && foreignAmountIn == 0) ||
                (nativeAmountIn == 0 && foreignAmountIn > 0),
            "BasePool::swap: Only One-Sided Swaps Supported"
        );
        (uint112 nativeReserve, uint112 foreignReserve, ) = getReserves(); // gas savings

        uint256 nativeBalance;
        uint256 foreignBalance;
        uint256 nativeAmountOut;
        uint256 foreignAmountOut;
        {
            // scope for _token{0,1}, avoids stack too deep errors
            IERC20 _nativeAsset = nativeAsset;
            IERC20 _foreignAsset = foreignAsset;
            nativeBalance = _nativeAsset.balanceOf(address(this));
            foreignBalance = _foreignAsset.balanceOf(address(this));

            require(
                to != address(_nativeAsset) && to != address(_foreignAsset),
                "BasePool::swap: Invalid Receiver"
            );

            if (foreignAmountIn > 0) {
                require(
                    foreignAmountIn <= foreignBalance - foreignReserve,
                    "BasePool::swap: Insufficient Tokens Provided"
                );
                require(
                    foreignAmountIn <= foreignReserve,
                    "BasePool::swap: Unfavourable Trade"
                );

                nativeAmountOut = VaderMath.calculateSwap(
                    foreignAmountIn,
                    foreignReserve,
                    nativeReserve
                );

                require(
                    nativeAmountOut > 0 && nativeAmountOut <= nativeReserve,
                    "BasePool::swap: Swap Impossible"
                );

                _nativeAsset.safeTransfer(to, nativeAmountOut); // optimistically transfer tokens
            } else {
                require(
                    nativeAmountIn <= nativeBalance - nativeReserve,
                    "BasePool::swap: Insufficient Tokens Provided"
                );
                require(
                    nativeAmountIn <= nativeReserve,
                    "BasePool::swap: Unfavourable Trade"
                );

                foreignAmountOut = VaderMath.calculateSwap(
                    nativeAmountIn,
                    nativeReserve,
                    foreignReserve
                );

                require(
                    foreignAmountOut > 0 && foreignAmountOut <= foreignReserve,
                    "BasePool::swap: Swap Impossible"
                );

                _foreignAsset.safeTransfer(to, foreignAmountOut); // optimistically transfer tokens
            }

            nativeBalance = _nativeAsset.balanceOf(address(this));
            foreignBalance = _foreignAsset.balanceOf(address(this));
        }

        _update(nativeBalance, foreignBalance, nativeReserve, foreignReserve);

        emit Swap(
            msg.sender,
            nativeAmountIn,
            foreignAmountIn,
            nativeAmountOut,
            foreignAmountOut,
            to
        );

        return nativeAmountOut > 0 ? nativeAmountOut : foreignAmountOut;
    }
```

contracts/mocks/MockUniswapV2Router.sol #349-367

```
    function _swap(
        uint256[] memory amounts,
        address[] memory path,
        address _to
    ) internal virtual {
        for (uint256 i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0, ) = UniswapV2Library.sortTokens(input, output);
            uint256 amountOut = amounts[i + 1];
            (uint256 amount0Out, uint256 amount1Out) = input == token0
                ? (uint256(0), amountOut)
                : (amountOut, uint256(0));
            address to = i < path.length - 2
                ? UniswapV2Library.pairFor(factory, output, path[i + 2])
                : _to;
            IUniswapV2Pair(UniswapV2Library.pairFor(factory, input, output))
                .swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

16. 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 #116-149

```
    function calculateSwapReverse(
        uint256 amountOut,
        uint256 reserveIn,
        uint256 reserveOut
    ) internal pure returns (uint256 amountIn) {
        // X * Y
        uint256 XY = reserveIn * reserveOut;

        // 2y
        uint256 y2 = amountOut * 2;

        // 4y
        uint256 y4 = y2 * 2;

        require(
            y4 < reserveOut,
            "VaderMath::calculateSwapReverse: Desired Output Exceeds Maximum Output Possible (1/4 of L
        );

        // root(-X^2 * Y * (4y - Y))     =>     root(X^2 * Y * (Y - 4y)) as Y - 4y >= 0     =>     Y >= 4y
        uint256 numeratorA = root(XY) * root(reserveIn * (reserveOut - y4));

        // X * (2y - Y)     =>     2yX - XY
        uint256 numeratorB = y2 * reserveIn;
        uint256 numeratorC = XY;

        // -1 * (root(-X^2 * Y * (4y - Y)) + (X * (2y - Y)))     =>     -1 * (root(X^2 * Y * (Y - 4y)) +
        uint256 numerator = numeratorC - numeratorA - numeratorB;

        // 2y
        uint256 denominator = y2;

        amountIn = numerator / denominator;
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

17. 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 #122-149

```
function addLiquidity(
    IERC20 tokenA,
    IERC20 tokenB,
    uint256 amountADesired,
    uint256 amountBDesired,
    address to,
    uint256 deadline
)
    public
    override
    ensure(deadline)
    returns (
        uint256 amountA,
        uint256 amountB,
        uint256 liquidity
    )
{
    IVaderPool pool;
    (pool, amountA, amountB) = _addLiquidity(
        address(tokenA),
        address(tokenB),
        amountADesired,
        amountBDesired
    );
    tokenA.safeTransferFrom(msg.sender, address(pool), amountA);
    tokenB.safeTransferFrom(msg.sender, address(pool), amountB);
    liquidity = pool.mint(to);
}
```

Recommendation

Add slippage limit check when do liquidity-related operations.

18. 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 #393-437

```
    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.

19. 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 #393-437

```
    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
                );
        }
    }
```

contracts/dex/router/VaderRouter.sol #243-258

```
    function swapTokensForExactTokens(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external virtual ensure(deadline) returns (uint256 amountIn) {
        amountIn = calculateInGivenOut(amountOut, path);

        require(
            amountInMax >= amountIn,
            "VaderRouter::swapTokensForExactTokens: Large Trade Input"
        );

        _swap(amountIn, path, to);
    }
```

Recommendation

Add slippage limit check when do liquidity-related operations.

20. 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 #452-496

```
    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.

21. 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/lbt/LiquidityBasedTWAP.sol #150-189

```
    function _updateVaderPrice(
        IUniswapV2Pair pair,
        ExchangePair storage pairData,
        uint256 timeElapsed
    ) internal returns (uint256 currentLiquidityEvaluation) {
        bool isFirst = pair.token0() == vader;

        (uint256 reserve0, uint256 reserve1, ) = pair.getReserves();

        (uint256 reserveNative, uint256 reserveForeign) = isFirst
            ? (reserve0, reserve1)
            : (reserve1, reserve0);

        (
            uint256 price0Cumulative,
            uint256 price1Cumulative,
            uint256 currentMeasurement
        ) = UniswapV2OracleLibrary.currentCumulativePrices(address(pair));

        uint256 nativeTokenPriceCumulative = isFirst
            ? price0Cumulative
            : price1Cumulative;

        unchecked {
            pairData.nativeTokenPriceAverage = FixedPoint.uq112x112(
                uint224(
                    (nativeTokenPriceCumulative -
                        pairData.nativeTokenPriceCumulative) / timeElapsed
                )
            );
        }

        pairData.nativeTokenPriceCumulative = nativeTokenPriceCumulative;

        pairData.lastMeasurement = currentMeasurement;

        currentLiquidityEvaluation =
            (reserveNative * previousPrices[uint256(Paths.VADER)]) +
            (reserveForeign * getChainlinkPrice(pairData.foreignAsset));
    }
```

contracts/lbt/LiquidityBasedTWAP.sol #113-148

```
    function syncVaderPrice()
        public
        override
        returns (
            uint256[] memory pastLiquidityWeights,
            uint256 pastTotalLiquidityWeight
        )
    {
        uint256 _totalLiquidityWeight;
        uint256 totalPairs = vaderPairs.length;
        pastLiquidityWeights = new uint256[](totalPairs);
        pastTotalLiquidityWeight = totalLiquidityWeight[uint256(Paths.VADER)];

        for (uint256 i; i < totalPairs; ++i) {
            IUniswapV2Pair pair = vaderPairs[i];
            ExchangePair storage pairData = twapData[address(pair)];
            uint256 timeElapsed = block.timestamp - pairData.lastMeasurement;

            if (timeElapsed < pairData.updatePeriod) continue;

            uint256 pastLiquidityEvaluation = pairData.pastLiquidityEvaluation;
            uint256 currentLiquidityEvaluation = _updateVaderPrice(
                pair,
                pairData,
                timeElapsed
            );

            pastLiquidityWeights[i] = pastLiquidityEvaluation;

            pairData.pastLiquidityEvaluation = currentLiquidityEvaluation;

            _totalLiquidityWeight += currentLiquidityEvaluation;
        }

        totalLiquidityWeight[uint256(Paths.VADER)] = _totalLiquidityWeight;
    }
```

Recommendation

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

22. 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/lbt/LiquidityBasedTWAP.sol #353-383

```
    function _updateUSDVPrice(
        IERC20 foreignAsset,
        ExchangePair storage pairData,
        uint256 timeElapsed
    ) internal returns (uint256 currentLiquidityEvaluation) {
        (uint256 reserveNative, uint256 reserveForeign, ) = vaderPool
            .getReserves(foreignAsset);

        (
            uint256 nativeTokenPriceCumulative,
            ,
            uint256 currentMeasurement
        ) = vaderPool.cumulativePrices(foreignAsset);

        unchecked {
            pairData.nativeTokenPriceAverage = FixedPoint.uq112x112(
                uint224(
                    (nativeTokenPriceCumulative -
                        pairData.nativeTokenPriceCumulative) / timeElapsed
                )
            );
        }

        pairData.nativeTokenPriceCumulative = nativeTokenPriceCumulative;

        pairData.lastMeasurement = currentMeasurement;

        currentLiquidityEvaluation =
            (reserveNative * previousPrices[uint256(Paths.USDV)]) +
            (reserveForeign * getChainlinkPrice(address(foreignAsset)));
    }
```

contracts/lbt/LiquidityBasedTWAP.sol #316-351

```
    function syncUSDVPrice()
        public
        override
        returns (
            uint256[] memory pastLiquidityWeights,
            uint256 pastTotalLiquidityWeight
        )
    {
        uint256 _totalLiquidityWeight;
        uint256 totalPairs = usdvPairs.length;
        pastLiquidityWeights = new uint256[](totalPairs);
        pastTotalLiquidityWeight = totalLiquidityWeight[uint256(Paths.USDV)];

        for (uint256 i; i < totalPairs; ++i) {
            IERC20 foreignAsset = usdvPairs[i];
            ExchangePair storage pairData = twapData[address(foreignAsset)];
            uint256 timeElapsed = block.timestamp - pairData.lastMeasurement;

            if (timeElapsed < pairData.updatePeriod) continue;

            uint256 pastLiquidityEvaluation = pairData.pastLiquidityEvaluation;
            uint256 currentLiquidityEvaluation = _updateUSDVPrice(
                foreignAsset,
                pairData,
                timeElapsed
            );

            pastLiquidityWeights[i] = pastLiquidityEvaluation;

            pairData.pastLiquidityEvaluation = currentLiquidityEvaluation;

            _totalLiquidityWeight += currentLiquidityEvaluation;
        }

        totalLiquidityWeight[uint256(Paths.USDV)] = _totalLiquidityWeight;
    }
```

Recommendation

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

Medium risk (0)

No Medium risk vulnerabilities found here

Low risk (0)

No Low risk vulnerabilities found here

Informational (0)

No Informational vulnerabilities found here

Disclaimer

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

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

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

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

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

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS 70-2021-12-vader (3New-FD) (XOK-SP) (OK-FLP) (1Positive-UT) (1Negative-FR) 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 70-2021-12-vader (3New-FD) (XOK-SP) (OK-FLP) (1Positive-UT) (1Negative-FR) 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.