METATRUST

Security Assessment for

MetaDefender

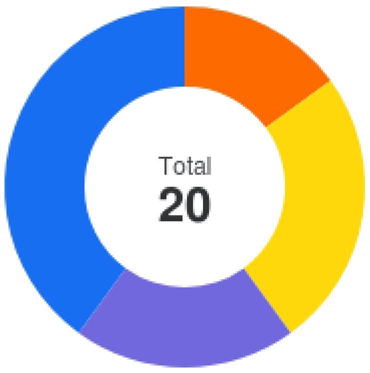
October 19, 2023

Executive Summary

Overview	
Project Name	MetaDefender
Codebase URL	https://github.com/Meta-Defender/metadefender-smart-contracts-v3
Scan Engine	Security Analyzer
Scan Time	2023/10/19 08:00:00
Commit Id	f99258d599cd2cbbda07c604aace505b1ebcac61 1c207b78aab54b8ad9de628da2235cf82780b8eb

Total	
Critical Issues	0
High risk Issues	3
Medium risk Issues	5
Low risk Issues	4
Informational Issues	8

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	15%	3
Medium risk Issues	25%	5
Low risk Issues	20%	4
Informational Issues	40%	8

Summary of Findings

MetaScan security assessment was performed on **October 19, 2023 08:00:00** on project **MetaDefender** 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 **20** vulnerabilities / security risks discovered during the scanning session, among which **0** critical vulnerabilities, **3** high risk vulnerabilities, **5** medium risk vulnerabilities, **4** low risk vulnerabilities, **8** informational issues.

ID	Description	Severity	Alleviation
MSA-001	Insecure LP Token Value Calculation	High risk	Mitigated
MSA-002	Wrong parameters used	High risk	Fixed
MSA-003	Proxy contracts are declared but implementation contracts are used to init contracts	High risk	Fixed
MSA-004	Potential Risk of the Usage of `tx.origin`	Medium risk	Fixed
MSA-005	Centralized Risk With Coin Transfer	Medium risk	Acknowledged
MSA-006	Centralized Risk With Key Variable Setting	Medium risk	Acknowledged
MSA-007	Invalid hardcoded contract address	Medium risk	Fixed
MSA-008	The modifier of the `epochCheck` function will malfunction	Medium risk	Fixed
MSA-009	Lack of invoking the `updateRewardDebtEpochIndex` function after claiming the reward	Low risk	Fixed
MSA-010	Wrong parameters in the event	Low risk	Fixed
MSA-011	`protocol` is always a zero address	Low risk	Fixed
MSA-012	A Test Contract is Used	Low risk	Fixed
MSA-013	Redundant state check of initialized	Informational	Fixed
MSA-014	Repeated Checks	Informational	Fixed
MSA-015	Unused variables	Informational	Fixed
MSA-016	Unused Functions	Informational	Mitigated
MSA-017	Unused imports	Informational	Fixed
MSA-018	Unused events	Informational	Acknowledged
MSA-019	Missing Event Setter	Informational	Mitigated
MSA-020	Mismatches between comments and functions	Informational	Acknowledged

Findings

Critical (0)

No Critical vulnerabilities found here

High risk (3)

1. Insecure LP Token Value Calculation

High risk   Security Analyzer

The function `getAseedPrice` calculates the `aseedPrice`, which depends on the `dex`. However, the price of the dex may be manipulated by malicious users.

File(s) Affected

contracts/EpochManage.sol #109-124

```
    function getAseedPrice() public view returns(uint){
        // aseed price feed logic
        address[] memory path1; //= [aseed,aca];
        path1[0] = aseed;
        path1[1] = aca;
        address[] memory path2; //= [aca,aseed];
        path2[0] = aca;
        path2[1] = aseed;
        uint aseed2aca1 = dex.getSwapTargetAmount(path1, 10**12);
        uint aseed2aca2 = dex.getSwapSupplyAmount(path2, 10**12);
        uint aseed2aca = aseed2aca1.add(aseed2aca2).div(2);

        uint acaPrice = oralce.getPrice(aca);
        uint aseedPrice = aseed2aca.mul(acaPrice).div(10**12);
        return aseedPrice;
    }
```

Recommendation

Recommend applying the TWAP(Time-weighted average price) to calculate the `aseedPrice`.

Alleviation Mitigated

The team applied the TWAP to mitigate this issue.

2. Wrong parameters used

High risk Security Analyzer

The third parameter of the `init` function for the `Policy` contract is `IEpochManage`. But, the third value passed to the `init` function when creating the `proxyPolicy` contract is `marketSet.mockRiskReserve`, which is wrong.

The same scenario happens when creating the `proxyEpochManage` contract, the parameters passed to the `init` function mismatch the definition of the `init` function of the `EpochManage` contract.

File(s) Affected

contracts/MetaDefenderFactory.sol #93-103

```
        ERC1967Proxy proxyPolicy = new ERC1967Proxy(
            address(marketSet.policy),
            abi.encodeWithSelector(
                Policy(address(0)).init.selector,
                marketSet.metaDefender,
                address(0),
                marketSet.mockRiskReserve,
                strConcat(_marketMessage.marketName, 'Policy'),
                strConcat(_marketMessage.marketSymbol, 'P')
            )
        );
```

contracts/MetaDefenderFactory.sol #112-120

```
        ERC1967Proxy proxyEpochManage = new ERC1967Proxy(
            address(marketSet.epochManage),
            abi.encodeWithSelector(
                EpochManage(address(0)).init.selector,
                marketSet.metaDefender,
                marketSet.liquidityCertificate,
                marketSet.policy
            )
        );
```

contracts/Policy.sol #39-45

```
    function init(
        address _metaDefender,
        address _protocol,
        IEpochManage _epochManage,
        string memory _name,
        string memory _symbol
    ) external initializer {
```

contracts/EpochManage.sol #62-69

```
    function init(
        IMetaDefender _metaDefender,
        ILiquidityCertificate _liquidityCertificate,
        IPolicy _policy,
        IOracle _oracle,
        IDEX _dex,
        address _oracleOperator
    ) external initializer {
```

Recommendation

Recommend fixing the parameter for creating the `proxyPolicy` contract.

Alleviation Fixed

The development team fixed this issue by removing the contract, in the commit 7f26cfcd42ce33a8adcb0197aad4698d8d5c8c91

3. Proxy contracts are declared but implementation contracts are used to init contracts

High risk Security Analyzer

The `deployMarkets` function creates some new contract instances, including `MetaDefender`, `LiquidityCertificate`, `Policy`, `MockRiskReserve`, and `EpochManage`, it also creates their proxy contracts, including `proxyMetaDefender`, `proxyLiquidityCertificate`, `proxyPolicy`, `mockRiskReserve`, and `proxyEpochManage`.

But, when initing the proxy contract `proxyMetaDefender`, the implementation contracts, `marketSet.mockRiskReserve`, `marketSet.liquidityCertificate`, `marketSet.policy`, and `marketSet.epochManage` are used, instead of the proxy contracts, `mockRiskReserve`, `proxyLiquidityCertificate`, `proxyPolicy`, and `proxyEpochManage`.

As a result, any state update of the proxy contracts, including `mockRiskReserve`, `proxyLiquidityCertificate`, `proxyPolicy`, and `proxyEpochManage`, will not affect the proxy contract `proxyMetaDefender`, which would result in errors.

Prove of Concept:

1. Creates two demo contracts `LiquidityCertificate` and `MetaDefender`;
2. Create the demo factory contract `MyFactory`;
3. In the `deploy` function of the `MyFactory` contract, create the proxy contract for `MetaDefender` with the `LiquidityCertificate` implementation contract and create the proxy contract for the `LiquidityCertificate` contract.
4. In the test script, call the `LiquidityCertificate` proxy contract's `setValue` function and update `value` to `1`. But the `value` in the `MetaDefender` proxy contract's member variable `lc` is still 0.

```
contract LiquidityCertificate {

    IERC20 public token;
    address public owner;
    uint public value;

    constructor() {
    }

    function init(IERC20 _token, address _owner) public {
        token = _token;
        owner = _owner;
    }

    function setValue(uint _value) public {
        value = _value;
    }
}

contract MetaDefender {

    LiquidityCertificate public lc;
    address public owner;

    constructor() {
    }

    function init(LiquidityCertificate _lc, address _owner) public {
        lc = _lc;
        owner = _owner;
    }
}

contract MyFactory {
    MetaDefender public md;
    LiquidityCertificate public lc;
    ERC1967Proxy public proxyMD;
    ERC1967Proxy public proxyLC;
    constructor() {
    }

    event log_keyvalue(string, address);
    event log_keyuint(string, uint);

    function deploy() public {
        md = new MetaDefender();
        // ERC20 token = new ERC20();
        lc = new LiquidityCertificate();
        emit log_keyvalue("md ", address(md));
        emit log_keyvalue("lc ", address(lc));

        proxyMD = new ERC1967Proxy(
            address(md),
            abi.encodeWithSelector(
                MetaDefender(address(0)).init.selector,
                lc, //init with the `LiquidityCertificate` implementaion contract
                msg.sender));
        emit log_keyvalue("proxyMD ", address(proxyMD));

        //new a proxy contract for LiquidityCertificate contract
        proxyLC = new ERC1967Proxy(
            address(lc),
            abi.encodeWithSelector(
                MetaDefender(address(0)).init.selector,
                lc,
                msg.sender));
        emit log_keyvalue("proxyLC ", address(proxyLC));
    }
}

//test
contract MyFactoryTest is Test {

    function testDeploy() public {
        MyFactory factory = new MyFactory();
        factory.deploy();

        bytes memory data;

        address proxyLC = address(factory.proxyLC());
        //1. call setValue function of proxyLC, the LiquidityCertificate proxy contract, to update value to 1
        address(proxyLC).call(
            abi.encodeWithSelector(LiquidityCertificate(address(0)).setValue.selector, 1));
        // read value
        (,  data) = address(proxyLC).call(
            abi.encodeWithSelector(LiquidityCertificate(address(0)).value.selector));
        uint proxyLC_value = abi.decode(data, (uint));//1

        //2. read the value of the MetaDefender proxy contract
        address proxyMD = address(factory.proxyMD());
        (, data) = address(proxyMD).call(
            abi.encodeWithSelector(MetaDefender(address(0)).lc.selector));
        LiquidityCertificate lc = abi.decode(data, (LiquidityCertificate));

        uint proxyMD_lc_value = lc.value();//0
        //3. values between the LiquidityCertificate proxy contract and the MetaDefender proxy contract are not the same
        assert(proxyLC_value != proxyMD_lc_value);
    }
}
```

The same scenario happens when initing proxy contracts, `proxyLiquidityCertificate`, `proxyPolicy`, `mockRiskReserve`, `proxyEpochManage`.

File(s) Affected

contracts/MetaDefenderFactory.sol #33-128

```
    function deployMarkets(
        MarketMessage memory _marketMessage,
        ITestERC20 _quoteToken,
        IAmericanBinaryOptions _americanBinaryOptions
    ) external onlyOwner {
        MetaDefender metaDefender = new MetaDefender();
        LiquidityCertificate liquidityCertificate = new LiquidityCertificate();
        Policy policy = new Policy();
        MockRiskReserve mockRiskReserve = new MockRiskReserve();
        EpochManage epochManage = new EpochManage();
        MarketSet memory marketSet = MarketSet(
            metaDefender,
            liquidityCertificate,
            policy,
            mockRiskReserve,
            epochManage
        );

        // then we try to init the markets.
        createProxyMarketSet(
            marketSet,
            _marketMessage,
            _quoteToken,
            _americanBinaryOptions
        );
    }

    function createProxyMarketSet(
        MarketSet memory marketSet,
        MarketMessage memory _marketMessage,
        ITestERC20 _quoteToken,
        IAmericanBinaryOptions _americanBinaryOptions
    ) internal {
        ERC1967Proxy proxyMetaDefender = new ERC1967Proxy(
            address(marketSet.metaDefender),
            abi.encodeWithSelector(
                MetaDefender(address(0)).init.selector,
                _quoteToken,
                owner(),
                owner(),
                marketSet.mockRiskReserve,
                marketSet.liquidityCertificate,
                marketSet.policy,
                _americanBinaryOptions,
                marketSet.epochManage,
                _marketMessage.initialRisk,
                _marketMessage.teamReserveRate,
                _marketMessage.standardRisk
            )
        );
        ERC1967Proxy proxyLiquidityCertificate = new ERC1967Proxy(
            address(marketSet.liquidityCertificate),
            abi.encodeWithSelector(
                LiquidityCertificate(address(0)).init.selector,
                marketSet.metaDefender,
                address(0),
                strConcat(_marketMessage.marketName, 'Certificate'),
                    strConcat(_marketMessage.marketSymbol, 'C')
                )
            );
            ERC1967Proxy proxyPolicy = new ERC1967Proxy(
                address(marketSet.policy),
                abi.encodeWithSelector(
                    Policy(address(0)).init.selector,
                    marketSet.metaDefender,
                    address(0),
                    marketSet.mockRiskReserve,
                     strConcat(_marketMessage.marketName, 'Policy'),
                     strConcat(_marketMessage.marketSymbol, 'P')
                 )
             );
             ERC1967Proxy mockRiskReserve = new ERC1967Proxy(
                 address(marketSet.mockRiskReserve),
                 abi.encodeWithSelector(
                     MockRiskReserve(address(0)).init.selector,
                     marketSet.metaDefender,
                     _quoteToken
                 )
             );
             ERC1967Proxy proxyEpochManage = new ERC1967Proxy(
                 address(marketSet.epochManage),
                 abi.encodeWithSelector(
                     EpochManage(address(0)).init.selector,
                     marketSet.metaDefender,
                     marketSet.liquidityCertificate,
                     marketSet.policy
                 )
             );
             emit MetaDefenderProxyDeployed(address(proxyMetaDefender));
             emit LiquidityCertificateProxyDeployed(
                 address(proxyLiquidityCertificate)
             );
             emit PolicyProxyDeployed(address(proxyPolicy));
             emit MockRiskReserveProxyDeployed(address(proxyPolicy));
             emit EpochManageProxyDeployed(address(proxyPolicy));
         }
```

Recommendation

Recommend initing contracts with declared proxy contracts or providing interfaces to update the contract address.

Example:

```
contract MyFactory {
    MetaDefender public md;
    LiquidityCertificate public lc;
    ERC1967Proxy public proxyMD;
    ERC1967Proxy public proxyLC;
    constructor() {
    }

    event log_keyvalue(string, address);
    event log_keyuint(string, uint);

    function deploy2() public {
        md = new MetaDefender();
        // ERC20 token = new ERC20();
        lc = new LiquidityCertificate();
        emit log_keyvalue("md ", address(md));
        emit log_keyvalue("lc ", address(lc));

        proxyLC = new ERC1967Proxy(
            address(lc),
            abi.encodeWithSelector(
                MetaDefender(address(0)).init.selector,
                lc,
                msg.sender));
        emit log_keyvalue("proxyLC ", address(proxyLC));

        proxyMD = new ERC1967Proxy(
            address(md),
            abi.encodeWithSelector(
                MetaDefender(address(0)).init.selector,
                address(proxyLC),//init with the `LiquidityCertificate` proxy contract
                msg.sender));
        emit log_keyvalue("proxyMD ", address(proxyMD));
    }
}

//test
contract MyFactoryTest is Test {

    function testDeploy2() public {
        MyFactory factory = new MyFactory();
        factory.deploy2();
        bytes memory data;
        address proxyLC = address(factory.proxyLC());
        address(proxyLC).call(
            abi.encodeWithSelector(LiquidityCertificate(address(0)).setValue.selector, 1));

        (,  data) = address(proxyLC).call(
            abi.encodeWithSelector(LiquidityCertificate(address(0)).value.selector));
        uint proxyLC_l = abi.decode(data, (uint));//1

        address proxyMD = address(factory.proxyMD());
        (, data) = address(proxyMD).call(
            abi.encodeWithSelector(MetaDefender(address(0)).lc.selector));
        LiquidityCertificate lc = abi.decode(data, (LiquidityCertificate));

        uint proxyMD_lc_l = lc.value();//1

        assert(proxyLC_l == proxyMD_lc_l);
        }
}
```

Alleviation Fixed

The development team fixed this issue by removing the contract, in the commit 7f26cfcd42ce33a8adcb0197aad4698d8d5c8c91

Medium risk (5)

1. Potential Risk of the Usage of `tx.origin`

Medium risk Security Analyzer

It is not recommended to use `tx.origin` for authorization. `tx.origin` based protection can be abused by a malicious contract if a legitimate user interacts with the malicious contract by mistake. Especially, phishing websites always take advantage of the `tx.origin` value to attack the victims.

File(s) Affected

contracts/LiquidityCertificate.sol #186-202

```
    function decreaseLiquidity(
        uint256 certificateId,
        bool isForce
    ) external override onlyMetaDefender {
        if (!isForce) {
            require(
                _isApprovedOrOwner(tx.origin, certificateId),
                'attempted to expire nonexistent certificate, or not owner'
            );
        }
        totalValidCertificateLiquidity = totalValidCertificateLiquidity.sub(
            _certificateInfo[certificateId].liquidity
        );
        totalPendingCertificateLiquidity = totalPendingCertificateLiquidity.sub(
            _certificateInfo[certificateId].liquidity
        );
    }
```

contracts/LiquidityCertificate.sol #210-224

```
    function expire(
        uint256 certificateId,
        uint64 currentEpochIndex,
        bool isForce
    ) external override onlyMetaDefender {
        if (!isForce) {
            require(
                _isApprovedOrOwner(tx.origin, certificateId),
                'attempted to expire nonexistent certificate, or not owner'
            );
        }
        _certificateInfo[certificateId].exitedEpochIndex = currentEpochIndex;
        _certificateInfo[certificateId].isValid = false;
        emit Expired(certificateId);
    }
```

Recommendation

Recommend passing the `msg.sender` as parameter to the invoked functions instead of using the `tx.origin`.

Alleviation Fixed

The team fixed this issue by removing related codes.

2. Centralized Risk With Coin Transfer

Medium risk Security Analyzer

The privileged role `official` can transfer the token `aUSD` to the account `official`.

File(s) Affected

contracts/MetaDefender.sol #154-160

```
    function teamClaim() external override {
        if (msg.sender != official) {
            revert InsufficientPrivilege();
        }
        aUSD.transfer(official, globalInfo.reward4Team);
        globalInfo.reward4Team = 0;
    }
```

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 Acknowledged

The team acknowledged this issue and will fix it in the future.

3. Centralized Risk With Key Variable Setting

Medium risk   Security Analyzer

In the `EpochManage` contract, the privileged role `oracleOperator` can invoke the following functions:

- The `feedPrice` function to update the `isAboveStike` and `daysAboveStrikePrice`;
- The `setStrikePrice` function to update the `strikePriceSetted` and the `strikePrice`.

In the `MetaDefenderMarketsRegistry` contract, the privileged role `owner` can invoke the following functions:

- The `addMarket` function to add a market;
- The `removeMarket` function to remove a market.

In the `MetaDefenderFactory` contract, the privileged role `owner` can invoke the following functions:

- The `deployMarkets` function to deploy a market.

File(s) Affected

contracts/periphery/MetaDefenderMarketsRegistry.sol #28-63

```
    function addMarket(
        address metaDefender,
        address liquidityCertificate,
        address policy,
        address epochManage,
        string memory marketName,
        string memory marketDescription,
        string memory marketPaymentToken,
        string memory marketProtectionType,
        string memory network
    ) external onlyOwner {
        require(insuranceMarkets.add(metaDefender), 'market already present');
        insuranceMarketsAddresses[metaDefender] = MarketAddresses(
            liquidityCertificate,
            policy,
            epochManage
        );
        insuranceMarketsMessages[metaDefender] = MarketMessages(
            marketName,
            marketDescription,
            marketPaymentToken,
            marketProtectionType,
            network
        );
        emit MarketAdded(
            metaDefender,
            liquidityCertificate,
            policy,
            epochManage,
            marketName,
            marketDescription,
            marketPaymentToken,
            marketProtectionType,
            network
        );
    }
```

contracts/periphery/MetaDefenderMarketsRegistry.sol #70-76

```
    function removeMarket(address metaDefender) external onlyOwner {
        require(insuranceMarkets.remove(metaDefender), 'market not present');
        delete insuranceMarketsAddresses[metaDefender];
        delete insuranceMarketsMessages[metaDefender];

        emit MarketRemoved(metaDefender);
    }
```

contracts/MetaDefender.sol #753-757

```
    function epochCheck() external override checkNewEpoch {
        require(msg.sender == official && !manuallyChecked);
        require(block.timestamp > epochManage.startTime()+epochManage.optionTradeableDuration());
        manuallyChecked = true;
    }
```

contracts/MetaDefenderFactory.sol #33-58

```
    function deployMarkets(
        MarketMessage memory _marketMessage,
        ITestERC20 _quoteToken,
        IAmericanBinaryOptions _americanBinaryOptions
    ) external onlyOwner {
        MetaDefender metaDefender = new MetaDefender();
        LiquidityCertificate liquidityCertificate = new LiquidityCertificate();
        Policy policy = new Policy();
        MockRiskReserve mockRiskReserve = new MockRiskReserve();
        EpochManage epochManage = new EpochManage();
        MarketSet memory marketSet = MarketSet(
            metaDefender,
            liquidityCertificate,
            policy,
            mockRiskReserve,
            epochManage
        );

        // then we try to init the markets.
        createProxyMarketSet(
            marketSet,
            _marketMessage,
            _quoteToken,
            _americanBinaryOptions
        );
    }
```

contracts/EpochManage.sol #84-100

```
    function feedPrice() external {
        require(msg.sender == oracleOperator && isWithdrawDay()); //start hosting price change after tl
        if(daysAboveStrikePrice >= 8){ // 7 days above strike price,then execisable, no need to feed ai
            return;
        }
        if(getAseedPrice()>=strikePrice){
            if(isAboveStike[getCurrentEpoch()] == true){
                return;
            }else{
                isAboveStike[getCurrentEpoch()] = true;
                daysAboveStrikePrice+=1;
            }
        }else{
            isAboveStike[getCurrentEpoch()] = false;
            daysAboveStrikePrice = 0; //once below strike price, restart
        }
    }
```

contracts/EpochManage.sol #103-107

```
    function setStrikePrice(uint _price) external {
        require(msg.sender == oracleOperator && strikePriceSetted == false && isWithdrawDay());
        strikePriceSetted = true;
        strikePrice = _price;
    }
```

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 Acknowledged

The team acknowledged this issue and will fix it in the future.

4. Invalid hardcoded contract address

Medium risk Security Analyzer

In the commit `71a1e077b39ca5557f02bcbc1de6d9ddd366acfa`, both the contract `Price` and the contract `EpochManage` use hardcoded contract addresses, which are invalid on-chain:

```
    address constant ORACLE = 0x0000000000000000000000000000000000000801;
    address constant DEX = 0x0000000000000000000000000000000000000803;

    IOracle oracle = IOracle(ORACLE);
    IDEX dex = IDEX(DEX);
```

File(s) Affected

EpochManage.sol #47-48

```
    address constant ORACLE = 0x0000000000000000000000000000000000000801;
    address constant DEX = 0x0000000000000000000000000000000000000803;
```

Prices.sol #18-19

```
    address constant ORACLE = 0x0000000000000000000000000000000000000801;
    address constant DEX = 0x0000000000000000000000000000000000000803;
```

Recommendation

Recommend removing the hardcoded contract address and initializing them in the constructor.

Alleviation Fixed

The team fixed this issue by initializing them in the constructor, in commits 60f40e155921e852f2abe3fe871f5fbef09aa38c and b40a3bf8c61b673eb414e3ee313dc16a724b502d.

5. The modifier of the **`epochCheck`** function will malfunction

Medium risk Security Analyzer

The **`epochCheck`** function has a modifier **`checkNewEpoch`** and it updates the **`manuallyChecked`** to true, which results in the modifier will not invoke the **`epochManage.checkAndCreateNewEpochAndUpdateAccRPSAccSPS();`**. It could result in unexpected results.

File(s) Affected

contracts/MetaDefender.sol #753-768

```
    function epochCheck() external override checkNewEpoch {
        require(msg.sender == official && !manuallyChecked);
        require(block.timestamp > epochManage.startTime()+epochManage.optionTradeableDuration());
        manuallyChecked = true;
    }

    modifier checkNewEpoch() virtual {
        if(!manuallyChecked){
            epochManage.checkAndCreateNewEpochAndUpdateLiquidity();
            }
        _;
        if(!manuallyChecked){
            epochManage.checkAndCreateNewEpochAndUpdateAccRPSAccSPS();
            }

    }
```

Recommendation

Recommend checking if it is an intended design and refining it if necessary.

Alleviation Fixed

The team fixed this issue by invoking the `checkAndCreateNewEpochAndUpdateAccRPSAccSPS` function at the `epochCheck` function.

Low risk (4)

1. Lack of invoking the `updateRewardDebtEpochIndex` function after claiming the reward

Low risk   Security Analyzer

In the `claimRewards` function, the `updateRewardDebtEpochIndex` function is correctly invoked to update the `rewardDebtEpochIndex` after transferring the reward. But, in the `certificateProviderExit` function, there is no invoking on the `updateRewardDebtEpochIndex` after transferring the reward, which would result in unexpected result.

File(s) Affected

contracts/MetaDefender.sol #351-364

```
        if (isForce) {
            if (rewards > 0) {
                aUSD.transfer(
                    liquidityCertificate.belongsTo(certificateId),
                    rewards
                );
            }
        } else {
            uint256 fee = withdrawal.multiplyDecimal(WITHDRAWAL_FEE_RATE);
            globalInfo.reward4Team = globalInfo.reward4Team.add(fee);
            if (withdrawal.add(rewards) > 0) {
                aUSD.transfer(msg.sender, withdrawal.add(rewards).sub(fee));
            }
        }
```

Recommendation

Recommend invoking the `updateRewardDebtEpochIndex` function after the transferring reward for the `certificateProviderExit` function.

Alleviation Fixed

The team fixed this issue by removing the if branch for checking the `isForce`.

2. Wrong parameters in the event

Low risk Security Analyzer

When emitting events, `MockRiskReserveProxyDeployed` and `EpochManageProxyDeployed`, in the `createProxyMarketSet` function, their parameters are wrong.

The same scenario happens for the event `ProviderExit` in the `certificateProviderExit` function of the `MetaDefender` contract, when the parameter `isForce` is true, the parameter passed to the `ProviderExit` event should be `liquidityCertificate.belongsTo(certificateId)`.

File(s) Affected

contracts/MetaDefenderFactory.sol #126-127

```
        emit MockRiskReserveProxyDeployed(address(proxyPolicy));
        emit EpochManageProxyDeployed(address(proxyPolicy));
```

contracts/MetaDefender.sol #351-365

```
        if (isForce) {
            if (rewards > 0) {
                aUSD.transfer(
                    liquidityCertificate.belongsTo(certificateId),
                    rewards
                );
            }
        } else {
            uint256 fee = withdrawal.multiplyDecimal(WITHDRAWAL_FEE_RATE);
            globalInfo.reward4Team = globalInfo.reward4Team.add(fee);
            if (withdrawal.add(rewards) > 0) {
                aUSD.transfer(msg.sender, withdrawal.add(rewards).sub(fee));
            }
        }
        emit ProviderExit(msg.sender);
```

Recommendation

Recommend updating the event parameters.

Alleviation Fixed

The team fixed this issue by removing the contract.

3. `protocol` is always a zero address

Low risk

Security Analyzer

The proxy contract `proxyLiquidityCertificate` initializes the implementation contract with the parameter `protocol` as address(0). Meanwhile, there is no function for the implementation contract to update the `protocol`, which results in the `protocol` will always be `address(0)`. It is meaningless.

The same scenario happens when creating the `proxyPolicy`.

File(s) Affected

contracts/LiquidityCertificate.sol #54-54

```
        protocol = _protocol;
```

contracts/MetaDefenderFactory.sol #83-92

```
        ERC1967Proxy proxyLiquidityCertificate = new ERC1967Proxy(
            address(marketSet.liquidityCertificate),
            abi.encodeWithSelector(
                LiquidityCertificate(address(0)).init.selector,
                marketSet.metaDefender,
                address(0),
                strConcat(_marketMessage.marketName, 'Certificate'),
                strConcat(_marketMessage.marketSymbol, 'C')
            )
        );
```

contracts/MetaDefenderFactory.sol #93-103

```
        ERC1967Proxy proxyPolicy = new ERC1967Proxy(
            address(marketSet.policy),
            abi.encodeWithSelector(
                Policy(address(0)).init.selector,
                marketSet.metaDefender,
                address(0),
                marketSet.mockRiskReserve,
                strConcat(_marketMessage.marketName, 'Policy'),
                strConcat(_marketMessage.marketSymbol, 'P')
            )
        );
```

Recommendation

Recommend checking the design and refining the logic accordingly.

Alleviation Fixed

The team fixed this issue by removing the factory contract and initializing the contracts manually.

4. A Test Contract is Used

Low risk   Security Analyzer

A mock contract **`MockRiskReserve`** is used and initialized in the **`MetaDefenderFactory`** contract, which could result in unexpected results.

File(s) Affected

contracts/MetaDefenderFactory.sol #8-8

```
import './Test-helpers/MockRiskReserve.sol';
```

contracts/MetaDefenderFactory.sol #41-49

```
        MockRiskReserve mockRiskReserve = new MockRiskReserve();
        EpochManage epochManage = new EpochManage();
        MarketSet memory marketSet = MarketSet(
            metaDefender,
            liquidityCertificate,
            policy,
            mockRiskReserve,
            epochManage
        );
```

contracts/MetaDefenderFactory.sol #104-111

```
        ERC1967Proxy mockRiskReserve = new ERC1967Proxy(
            address(marketSet.mockRiskReserve),
            abi.encodeWithSelector(
                MockRiskReserve(address(0)).init.selector,
                marketSet.metaDefender,
                _quoteToken
            )
        );
```

Recommendation

Recommend replacing the mock contract with the real contract.

Alleviation Fixed

The team fixed this issue by removing the contract.

Informational (8)

1. Redundant state check of initialized

Informational Security Analyzer

The `init` functions of the `Policy` contract and the `LiquidityCertificate` contract not only use the `initializer` modifier to prevent them from being initialized twice but also declare and check a state variable `initialized` to do the same thing.

The `initializer` modifier is good enough and there is no need to do the same thing with declaring and checking a state variable `initialized`.

Prove of concept:

1. Create a contract `Initializer` with an `init` function, which has a modifier `initializer`. There is a state variable counter to count the times of calling the `init` function.
2. Create a proxy contract with the `Initializer` contract as the implementation contract and invoke the `init` function twice. The result is that the second call of the `init` function failed and the `counter` is 1.

```
contract Initializer is ERC721EnumerableUpgradeable {

    uint public count;
    function init() public initializer {
        count++;
    }
}

//test
contract InitializerTest is Test {

    function testDeploy() public {
        Initializer myInitilizer = new Initializer();

        ERC1967Proxy proxy = new ERC1967Proxy(
            address(myInitilizer),
            abi.encodeWithSelector(
                Initializer(address(0)).init.selector)
        );
        (bool ret, bytes memory data) = address(proxy).call(
            abi.encodeWithSelector(Initializer(address(0)).init.selector));
        assert(ret == false);
        (, data) = address(proxy).call(
            abi.encodeWithSelector(Initializer(address(0)).count.selector));
        uint count = abi.decode(data, (uint));
        assert(count == 1);
    }
}
```

File(s) Affected

contracts/Policy.sol #39-56

```
    function init(
        address _metaDefender,
        address _protocol,
        IEpochManage _epochManage,
        string memory _name,
        string memory _symbol
    ) external initializer {
        require(!initialized, 'already initialized');
        require(
            _metaDefender != address(0),
            'liquidityPool cannot be 0 address'
        );
        metaDefender = _metaDefender;
        protocol = _protocol;
        epochManage = _epochManage;
        __ERC721_init(_name, _symbol);
        initialized = true;
    }
```

contracts/LiquidityCertificate.sol #42-57

```
    function init(
        address _metaDefender,
        address _protocol,
        string memory _name,
        string memory _symbol
    ) external initializer {
        require(
            _metaDefender != address(0),
            'liquidityPool cannot be 0 address'
        );
        require(!initialized, 'already initialized');
        metaDefender = _metaDefender;
        protocol = _protocol;
        __ERC721_init(_name, _symbol);
        initialized = true;
    }
```

Recommendation

Recommend removing repeatedly checks for the initialized state.

Alleviation Fixed

The team fixed this issue by removing the modifier `initializer`, in commit 2575a22918aee1369535ab0070777399f03293c7.

2. Repeated Checks

Informational Security Analyzer

In the **burn** function, there is a modifier **onlyMetaDefender**, that validate caller:

```
    modifier onlyMetaDefender() virtual {
        require(msg.sender == address(metaDefender), 'Only MetaDefender');
        _;
    }

    function burn(
        address spender,
        uint256 policyId
    ) external override onlyMetaDefender {
        if (msg.sender != metaDefender) {
            revert InsufficientPrivilege();
        }
```

Thus, there is no need to repeatedly validate **msg.sender** in the function's body.

The same scenario happens in the **changeStatusIsClaimed** function and the **changeStatusIsClaimApplying** function.

File(s) Affected

contracts/Policy.sol #225-231

```
    function burn(
        address spender,
        uint256 policyId
    ) external override onlyMetaDefender {
        if (msg.sender != metaDefender) {
            revert InsufficientPrivilege();
        }
```

contracts/Policy.sol #290-296

```
    function changeStatusIsClaimApplying(
        uint256 policyId,
        bool status
    ) external override onlyMetaDefender {
        if (msg.sender != metaDefender) {
            revert InsufficientPrivilege();
        }
```

contracts/Policy.sol #250-256

```
    function changeStatusIsClaimed(
        uint256 policyId,
        bool status
    ) external override onlyMetaDefender {
        if (msg.sender != metaDefender) {
            revert InsufficientPrivilege();
        }
```

Recommendation

Recommend removing the redundant validations in the functions' body.

Alleviation Fixed

The team fixed this issue by removing the redundant check, in commit 1c207b78aab54b8ad9de628da2235cf82780b8eb.

3. Unused variables

Informational   Security Analyzer

Variables, `protocol` and `BUFFER` are unused.

File(s) Affected

contracts/MetaDefender.sol #51-51

```
address public protocol;
```

contracts/MetaDefender.sol #55-55

```
uint256 public constant MAX_COVERAGE_PERCENTAGE = 2e17;
```

contracts/MetaDefender.sol #57-57

```
uint256 public constant BUFFER = 3;
```

Recommendation

Recommend removing the redundant variables.

Alleviation Fixed

The team fixed this issue by removing the redundant variable, in commit 7f26cfcd42ce33a8adcb0197aad4698d8d5c8c91.

4. Unused Functions

Informational Security Analyzer

The `burn` function of the `Policy` contract can only be invoked by the `metaDefender` since the `burn` function has the modifier `onlyMetaDefender`.

```
    modifier onlyMetaDefender() virtual {
        require(msg.sender == address(metaDefender), 'Only MetaDefender');
        _;
    }

    function burn(
        address spender,
        uint256 policyId
    ) external override onlyMetaDefender {
```

But, the `MetaDefender` contract never calls the `burn` function of the `Policy` contract.

The function `claimPolicy(uint256 policyId,bool isReserve) internal` is also unused.

File(s) Affected

contracts/Policy.sol #225-228

```
    function burn(
        address spender,
        uint256 policyId
    ) external override onlyMetaDefender {
```

contracts/MetaDefender.sol #593-596

```
    function claimPolicy(
        uint256 policyId,
        bool isReserve
    ) internal  {
```

Recommendation

Recommend checking the `Policy` contract and the `MetaDefender` contract for the `burn` function, redesigning their logic if necessary, and removing other unused functions.

Alleviation Mitigated

The team partially fixed this issue by removing the redundant function `claimPolicy(uint256 policyId,bool isReserve)`.

5. Unused imports

Informational Security Analyzer

The import of `hardhat/console.sol` is unused and is only for testing.

File(s) Affected

contracts/MetaDefender.sol #5-5

```
import 'hardhat/console.sol';
```

contracts/periphery/GlobalsViewer.sol #12-12

```
import 'hardhat/console.sol';
```

Recommendation

Recommend removing redundant imports.

Alleviation Fixed

The development team fixed this issue in the commit 7f26cfcd42ce33a8adcb0197aad4698d8d5c8c91

6. Unused events

Informational Security Analyzer

There are many events in the `MetaDefender` contract that are unused.

File(s) Affected

contracts/MetaDefender.sol #819-819

```
    error InvalidMiningProxy(address proxy);
```

contracts/MetaDefender.sol #801-802

```
    error InsufficientPrivilege();
    error InsufficientUsableCapital();
```

contracts/MetaDefender.sol #805-812

```
    error ProviderDetected(address providerAddress);
    error ProviderNotExist(uint256 _certificateId);
    error ProviderNotStale(uint256 id);
    error PolicyAlreadyCancelled(uint256 id);
    error PreviousPolicyNotCancelled(uint256 id);
    error PolicyCanNotBeCancelled(uint256 id);
    error PolicyCanOnlyCancelledByHolder(uint256 id);
    error InvalidPolicy(uint256 id);
```

Recommendation

Recommend removing unused events.

Alleviation Acknowledged

The team acknowledged this issue and will fix it in the future.

7. Missing Event Setter

Informational Security Analyzer

Functions that update the state variables are recommended to emit events

File(s) Affected

contracts/MetaDefender.sol #87-118

```
    function init(
        // basic information
        IERC20 _aUSD,
        address _judger,
        address _official,
        // riskReserve
        IMockRiskReserve _mockRiskReserve,
        // NFT LPs and policy NFT
        ILiquidityCertificate _liquidityCertificate,
        IPolicy _policy,
        // calculation
        IAmericanBinaryOptions _americanBinaryOptions,
        // functional contracts.
         IEpochManage _epochManage,
         // params
         uint256 _initialRisk,
         uint256 _teamReserveRate,
         uint256 _standardRisk
     ) external initializer {
         aUSD = _aUSD;
         judger = _judger;
         official = _official;
         mockRiskReserve = _mockRiskReserve;
         liquidityCertificate = _liquidityCertificate;
         policy = _policy;
         epochManage = _epochManage;
         americanBinaryOptions = _americanBinaryOptions;
         globalInfo.risk = _initialRisk;
         globalInfo.standardRisk = _standardRisk;
         teamReserveRate = _teamReserveRate;
         initialized = true;
     }
```

contracts/MetaDefender.sol #133-138

```
    function transferJudger(address _judger) external override {
        if (msg.sender != judger) {
            revert InsufficientPrivilege();
        }
        judger = _judger;
    }
```

contracts/MetaDefender.sol #144-149

```
    function transferOfficial(address _official) external override {
        if (msg.sender != official) {
            revert InsufficientPrivilege();
        }
        official = _official;
    }
```

contracts/MetaDefender.sol #154-160

```
    function teamClaim() external override {
        if (msg.sender != official) {
            revert InsufficientPrivilege();
        }
        aUSD.transfer(official, globalInfo.reward4Team);
        globalInfo.reward4Team = 0;
    }
```

contracts/MetaDefender.sol #182-187

```
    function updateStandardRisk(uint256 standardRisk) external override {
        if (msg.sender != official) {
            revert InsufficientPrivilege();
        }
        globalInfo.standardRisk = standardRisk;
    }
```

contracts/MetaDefender.sol #499-516

```
    function claimRewards(
        uint256 certificateId
    ) external override nonReentrant checkNewEpoch {
        if (msg.sender != (liquidityCertificate.belongsTo(certificateId))) {
            revert InsufficientPrivilege();
        }
        uint256 rewards = getRewards(certificateId, false);
        if (rewards > 0) {
            uint64 previousEpochIndex = epochManage.currentEpochIndex() - 1;
            liquidityCertificate.updateRewardDebtEpochIndex(
                certificateId,
                previousEpochIndex
            );
            aUSD.transfer(msg.sender, rewards);
        } else {
            revert NoRewards();
        }
    }
```

contracts/MetaDefender.sol #523-543

```
    function withdrawAfterExit(
        uint256 certificateId
    ) external override nonReentrant  {
        ILiquidityCertificate.CertificateInfo
            memory certificateInfo = liquidityCertificate.getCertificateInfo(
                certificateId
            );
        if (msg.sender != (liquidityCertificate.belongsTo(certificateId))) {
            revert InsufficientPrivilege();
        }
        if (certificateInfo.exitedEpochIndex == 0) {
            revert CertificateNotExit();
        }
        (uint256 SPSLocked, uint256 withdrawal) = getSPSLockedByCertificateId(
            certificateId
        );
        liquidityCertificate.updateSPSLocked(certificateId, SPSLocked);
        uint256 fee = withdrawal.multiplyDecimal(WITHDRAWAL_FEE_RATE);
        globalInfo.reward4Team = globalInfo.reward4Team.add(fee);
        aUSD.transfer(msg.sender, withdrawal.sub(fee));
    }
```

contracts/MetaDefender.sol #656-698

```
    function policyClaimApply(
        uint256 policyId
    ) external override  {
        // only for aseed option
        if(epochManage.daysAboveStrikePrice()<7){
            revert NotExerciseable();
        }
        //else option exercise allowed

        IPolicy.PolicyInfo memory policyInfo = policy.getPolicyInfo(policyId);
        IEpochManage.EpochInfo memory enteredEpochInfo = epochManage
            .getEpochInfo(policyInfo.enteredEpochIndex);
        IEpochManage.EpochInfo memory currentEpochInfo = epochManage
            .getCurrentEpochInfo();

        // in aseed option, once striked, exerciseable at any time
        // if (
        //     currentEpochInfo.epochId >
        //     enteredEpochInfo.epochId.add(policyInfo.duration).add(BUFFER)
        // ) {
        //     revert PolicyAlreadyStale(policyId);
        // }

        if (policyInfo.isClaimed == true) {
            revert PolicyAlreadyClaimed(policyId);
        }
        if (policyInfo.isSettled == true) { // kept although settle is not used in aseed option
            revert PolicyAlreadySettled(policyId);
        }
        if (policyInfo.beneficiary != msg.sender) {
            revert SenderNotBeneficiary(policyInfo.beneficiary, msg.sender);
        }
        if (policyInfo.isClaimApplying == true) { // kept although claimable or not is decided while ap
            revert ClaimUnderProcessing(policyId);
        }
        policy.changeStatusIsClaimApplying(policyId, true);

        //migrated from appove apply function for immediate exercise
        //claimPolicy(policyId, false);
        claimPolicy(policyId);
        aUSD.transfer(policyInfo.beneficiary, policyInfo.coverage);
        policy.changeStatusIsClaimed(policyId, true);
    }
```

contracts/MetaDefender.sol #753-757

```
    function epochCheck() external override checkNewEpoch {
        require(msg.sender == official && !manuallyChecked);
        require(block.timestamp > epochManage.startTime()+epochManage.optionTradeableDuration());
        manuallyChecked = true;
    }
```

Recommendation

Emit events in functions that update the state variable.

Alleviation Mitigated

The team partially fixed this issue by emitting events, in commit 7f26cfcd42ce33a8adcb0197aad4698d8d5c8c91.

8. Mismatches between comments and functions

Informational Security Analyzer

The comments of the functions `isClaimAvailable` and `isSettleAvailable` do not match the logic of their functions.

File(s) Affected

contracts/Policy.sol #161-167

```
    /**
     * @notice isCancelAvailable the to check if the policy can be cancelled now.
     * @param policyId The id of the policy.
     */
    function isSettleAvailable(
        uint256 policyId
    ) external view override returns (bool) {
```

contracts/Policy.sol #193-199

```
    /**
     * @notice isCancelAvailable the to check if the policy can be cancelled now.
     * @param policyId The id of the policy.
     */
    function isClaimAvailable(
        uint256 policyId
    ) external view override returns (bool) {
```

Recommendation

Recommend updating the comments on the functions.

Alleviation Acknowledged

The team acknowledged this issue and will fix it in the future.

Disclaimer

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.