Competitive Security Assessment

Mantle-LSD-mntEth

Nov 6th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

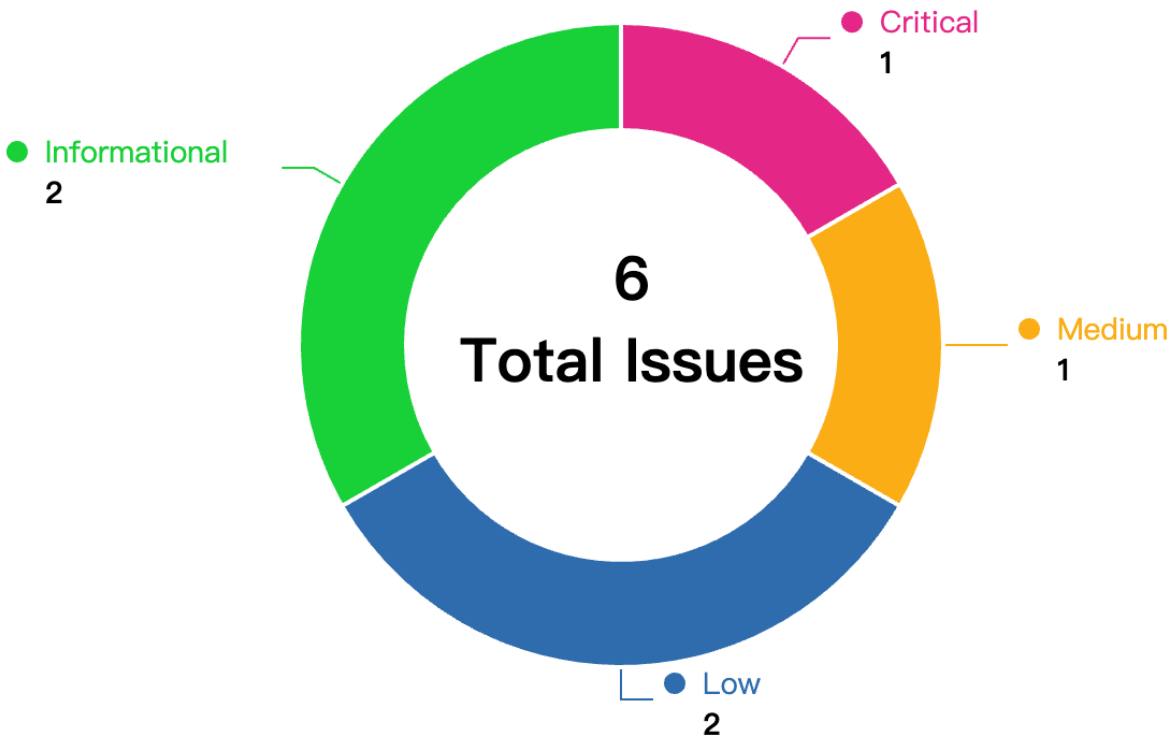
Project Name	Mantle-LSD-mntEth
Platform & Language	Solidity
Codebase	• https://github.com/TwoFiftySixLabs/mntEth • audit commit - 0649aaa6220689de155618d2ffa25999ed201f5b • final commit - 0649aaa6220689de155618d2ffa25999ed201f5b
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
src/Oracle.sol	cf3a365e8aba27d13e2b09d6458ef82d6233087170ed1923b634a626cf9a5e16
src/Staking.sol	be2ada0c6f3efab043ae23918b0dabec22767861df206ec36a6053e186b8ad9b
src/UnstakeRequestsManager.sol	ca71a3c9c9f303ec9bd2baaadd4d5940ec01a93abda4e904c0f8b8de8b6afa9b
src/ReturnsAggregator.sol	7be01f56ebca439b9904e84506b163e6146893885bc2dc262028ef706df491a1
src/OracleQuorumManager.sol	f5e5f38f1d27034cabc27ee688a495624ff88a011b5be0b3fdd7417329e1cf85
src/Pauser.sol	913a9a9b8035fb192fa2087a83035dce76ce820ff4f0851b454ce1a8ed683432
src/METH.sol	4a402260f805fa1d04d696e2dfdf53151c4534ab3a28fa0639352c2954698e9f
src/ReturnsReceiver.sol	5fc325fbeaebbf5ee36c3d7fcaac03ceffdbc84d529dc514b22cdf21d10e7df8
src/interfaces/IOracle.sol	ec3f3f47e13228364fff61f355b1a19eab5a6e9911ef22014be0a4c875d94a37
src/interfaces/IUnstakeRequestsManager.sol	0866a9d3ce3ccc3722aab39bb3a0a324ab073645d634df3eab88fe181ff1909f
src/interfaces/IDepositContract.sol	f13488e4f371bee428829e95cf294067bef29b20d7a9d5a4294c9e1941d9d07f
src/interfaces/IPauser.sol	0bb3520d8e7eea562b06abce6f3ccf236e533d36bea3aaafee8a941999f769f3
src/interfaces/IStaking.sol	727b93feef8cd737a582ca92d374541914138940820 4a4ea67137e22c5bc20f5
src/interfaces/IMETH.sol	319a1a4348f398456688cc90badf6f123157e70c417cdb61ec81a6ada33aef15
src/interfaces/IReturnsAggregator.sol	a74879c453baa1ac07c4a4444e20ae6da08a0c2a44100a5dca117699d15c8ef8

src/interfaces/ProtocolEvents.sol	**13a6d82e7fe246034e980d7ea4eef011ac5003524bd1df14bfd2eac3103800c1**

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
MNT-1	Unburned mETH in UnstakeRequestsManager decreases the value of mETHToETH	Logical	Critical	Acknowledged	thereksfour, infinityhacker, Kong7ych3
MNT-2	Incorrect calculation of `totalControlled()`	Logical	Medium	Mitigated	Yaodao
MNT-3	Missing Oracle Quorum Threshold Requirement and Dangerous Default Configuration	Oracle Manipulation	Medium	Declined	0x1337

MNT-4	Potential fail to pass `sanityCheckUpdate()` due to incorrect update the `minDepositPerValidator` and `maxDepositPerValidator`	Logical	Low	Acknowledged	Yaodao
MNT-5	`minimumUnstakeBound_` need more limit	Privilege Related	Low	Acknowledged	Satyam_
MNT-6	`min` variable should be less than `max` variable	Logical	Low	Declined	Yaodao, BradMoonUESTC
MNT-7	Missing Zero Address Check in `Staking::initialize` function	Code Style	Low	Declined	Xi_Zi
MNT-8	feesBasisPoints need more limit	Privilege Related	Low	Declined	Xi_Zi
MNT-9	Unlocked pragma	Language Specific	Informational	Acknowledged	rajatbeladiya
MNT-10	Missing Event Setter in `Staking::reclaimAllocatedETHSurplus` function	Code Style	Informational	Acknowledged	Xi_Zi

MNT-1:Unburned mETH in UnstakeRequestsManager decreases the value of mETHToETH

Category	Severity	Client Response	Contributor
Logical	Critical	Acknowledged	thereksfour, infinityhacker, Kong7ych3

Code Reference

- code/src/Staking.sol#L359-L380
- code/src/Staking.sol#L554-L570
- code/src/Staking.sol#L574-L585
- code/src/Staking.sol#L368
- code/src/Staking.sol#L377
- code/src/Staking.sol#L569
- code/src/Staking.sol#L529

```
359:function _unstakeRequest(uint128 methAmount, uint128 minETHAmount) internal returns (uint256) {
360:        if (pauser.isUnstakeRequestsAndClaimsPaused()) {
361:            revert Paused();
362:        }
363:
364:        if (methAmount < minimumUnstakeBound) {
365:            revert MinimumUnstakeBoundNotSatisfied();
366:        }
367:
368:        uint128 ethAmount = uint128(mETHToETH(methAmount));
369:        if (ethAmount < minETHAmount) {
370:            revert UnstakeBelowMinimumETHAmount(ethAmount, minETHAmount);
371:        }
372:
373:        uint256 requestID =
374:            unstakeRequestsManager.create({requester: msg.sender, mETHLocked: methAmount, ethReq
uested: ethAmount});
375:        emit UnstakeRequested({id: requestID, staker: msg.sender, ethAmount: ethAmount, mETHLock
ed: methAmount});
376:
377:        SafeERC20Upgradeable.safeTransferFrom(mETH, msg.sender, address(unstakeRequestsManager),
methAmount);
378:
379:        return requestID;
380:    }

368:uint128 ethAmount = uint128(mETHToETH(methAmount));

377:SafeERC20Upgradeable.safeTransferFrom(mETH, msg.sender, address(unstakeRequestsManager), methAmo
unt);

529:function ethToMETH(uint256 ethAmount) public view returns (uint256) {

554:function mETHToETH(uint256 mETHAmount) public view returns (uint256) {
555:        // 1:1 exchange rate on the first stake.
556:        // Using `METH.totalSupply` over `totalControlled` to check if the protocol is in its bo
otstrap phase since
557:        // the latter can be manipulated, for example by transferring funds to the `ExecutionLay
erReturnsReceiver`, and
558:        // therefore be non-zero by the time the first stake is made
559:        if (mETH.totalSupply() == 0) {
```

```
560:             return mETHAmount;
561:         }
562:
563:         // deltaETH = (totalControlled / mETHSupply) * mETHAmount
564:         // This rounds down to zero in the case of `mETHAmount * totalControlled < mETHSupply`.
565:         // While this scenario is theoretically possible, it can only be realised feasibly durin
g the protocol's
566:         // bootstrap phase and if `totalControlled` and `mETHSupply` can be changed independentl
y of each other. Since
567:         // the former is permissioned, and the latter is not permitted by the protocol, this can
not be exploited by an
568:         // attacker.
569:         return Math.mulDiv(mETHAmount, totalControlled(), mETH.totalSupply());
570:     }

569:return Math.mulDiv(mETHAmount, totalControlled(), mETH.totalSupply());

574:function totalControlled() public view returns (uint256) {
575:         OracleRecord memory record = oracle.latestRecord();
576:         uint256 total = 0;
577:         total += unallocatedETH;
578:         total += allocatedETHForDeposits;
579:         /// The total ETH deposited to the beacon chain must be decreased by the deposits proces
sed by the off-chain
580:         /// oracle since it will be accounted for in the currentTotalValidatorBalance from that
point onwards.
581:         total += totalDepositedInValidators - record.cumulativeProcessedDepositAmount;
582:         total += record.currentTotalValidatorBalance;
583:         total += unstakeRequestsManager.balance();
584:         return total;
585:     }
```

Description

thereksfour : mETHToETH() is used in _unstakeRequest() to calculate the amount of ETH the user can redeem, where the mETH/ETH exchange rate is totalControlled()/mETH.totalSupply()

```
    function _unstakeRequest(uint128 methAmount, uint128 minETHAmount) internal returns (uint256) {
        if (pauser.isUnstakeRequestsAndClaimsPaused()) {
            revert Paused();
        }

        if (methAmount < minimumUnstakeBound) {
            revert MinimumUnstakeBoundNotSatisfied();
        }

        uint128 ethAmount = uint128(mETHToETH(methAmount));
    ...
    function mETHToETH(uint256 mETHAmount) public view returns (uint256) {
        ...
        return Math.mulDiv(mETHAmount, totalControlled(), mETH.totalSupply());
    }
```

The problem here is that totalControlled() and mETH.totalSupply() include unburned mETH and unclaimed ETH in UnstakeRequestsManager, and since their exchange rates are fixed, including them in the calculation of the new exchange rate will result in the exchange rate being pulled down.

Consider the current totalControlled() = 110, mETH.totalSupply() = 100. Alice redeems 50 mETH, at which point the exchange rate is 110/100 = 1.1, and there will be 50 unburned mETH and 55 unclaimed ETH in the UnstakeRequestsManager. After some time, a reward of 10 ETH is added. If Bob redeems the other 50 mETH, the exchange rate at this point would be (110+10)/100 = 1.2, Bob's unclaimed reward would be 60 ETH, but Bob's true reward would be 110+10-55 = 65 ETH, i.e., the correct exchange rate would be (110+10-55)/(100-50) = 1.3.

It is worth noting that if there are a large number of users that don't claim their ETH and burn mETH in the UnstakeRequestsManager, due to the fact that these mETH and ETH are involved in the exchange rate calculations, this can significantly lower the mETH/ETH exchange rate.

infinityhacker : According to the `stake` and `unstakeRequest` function of `Staking` contract, they use `mETHToETH` and `ethToMETH` respectively, which both use `mETH.totalSupply()` internally. Both functions are aim to calculate the exchange rate of mETH to ETH. But because it use `mETH.totalSupply()` internally, it ignore those mETH in which already `ready-to-claim`, which makes the exchange rate incorrect.

Assume a malicious user, Alice, she had 50 mETH that has passed the claiming period and not claim yet, and the total supply of mETH is now 200, and `totalControlled()` is 250, according to the formula in `ethToMETH` function, she can first stake 50 ETH to roughly get `50 * 200 / 250 = 40` mETH, and then immediately claim 50 mETH and make a claim request. At this moment, `totalControlled()` is still 200 and according to the calculation in `mETHToETH`, Alice can get `50 * 200 / 150 = 66.66` ETH. so Alice did nothing but get 16.66 ETH reward, which is not expected.

Kong7ych3 : In the staking contract, when the user performs an unstakeRequest operation, it will first use the mETHToETH function to calculate the number of ETH tokens that the user can obtain based on the current totalControlled, and then transfer the mETH tokens to the unstakeRequestsManager contract without burning it. However, it should be noted that although the mETH tokens have been transferred to the unstakeRequestsManager contract, they have not yet been burned, so staking rewards will also be allocated to these mETH. Theoretically, after the user performs an unstakeRequest operation, the mETH in the unstakeRequestsManager contract should no longer be allocated to

rewards, and it should be excluded when calculating mETHToETH. Otherwise, this will cause other users to obtain lower than expected ethAmount when performing unstakeRequest operations.

Recommendation

thereksfour : It is recommended to burn the user's mETH directly when the user requests to redeem the mETH and record the corresponding amount of ETH as `redeemedETH`, and subtracting the `redeemedETH` in `totalControlled()`. Note that when the user claims the ETH in `UnstakeRequestsManager`, reduce the `redeemedETH`.
infinityhacker : Deduct mature claim request from calculation
Kong7ych3 : It is recommended that the total supply of mETH should be subtracted from the number of mETH tokens in the unstake state when performing mETHToETH calculations.

Client Response

Acknowledged.This is intentional by design for simplicity. We have performed extensive analysis and concluded that the effect on the rate is negligible at the scales the protocol will operate at. We will acknowledge rather than reject as the code was not documented to explain this case before the audit started.

MNT-2:Incorrect calculation of `totalControlled()`

Category	Severity	Client Response	Contributor
Logical	Medium	Mitigated	Yaodao

Code Reference

- code/src/Staking.sol#L574-L585
- code/src/ReturnsAggregator.sol#L110-L151

```
110:function processReturns(uint256 rewardAmount, uint256 principalAmount, bool shouldIncludeELRewar
ds)
111:        external
112:        assertBalanceUnchanged
113:    {
114:        if (msg.sender != address(oracle)) {
115:            revert NotOracle();
116:        }
117:
118:        // Calculate the total amount of returns that will be aggregated.
119:        uint256 clTotal = rewardAmount + principalAmount;
120:        uint256 totalRewards = rewardAmount;
121:
122:        uint256 elRewards = 0;
123:        if (shouldIncludeELRewards) {
124:            elRewards = address(executionLayerReceiver).balance;
125:            totalRewards += elRewards;
126:        }
127:
128:        // Calculate protocol fees.
129:        uint256 fees = Math.mulDiv(feesBasisPoints, totalRewards, _BASIS_POINTS_DENOMINATOR);
130:
131:        // Aggregate returns in this contract
132:        address payable self = payable(address(this));
133:        if (elRewards > 0) {
134:            executionLayerReceiver.transfer(self, elRewards);
135:        }
136:        if (clTotal > 0) {
137:            consensusLayerReceiver.transfer(self, clTotal);
138:        }
139:
140:        // Forward the net returns (if they exist) to the staking contract.
141:        uint256 netReturns = clTotal + elRewards - fees;
142:        if (netReturns > 0) {
143:            staking.receiveReturns{value: netReturns}();
144:        }
145:
146:        // Send protocol fees (if they exist) to the fee receiver wallet.
147:        if (fees > 0) {
148:            emit FeesCollected(fees);
149:            Address.sendValue(feesReceiver, fees);
```

```
150:         }
151:     }

574:function totalControlled() public view returns (uint256) {
575:         OracleRecord memory record = oracle.latestRecord();
576:         uint256 total = 0;
577:         total += unallocatedETH;
578:         total += allocatedETHForDeposits;
579:         /// The total ETH deposited to the beacon chain must be decreased by the deposits proces
sed by the off-chain
580:         /// oracle since it will be accounted for in the currentTotalValidatorBalance from that
point onwards.
581:         total += totalDepositedInValidators - record.cumulativeProcessedDepositAmount;
582:         total += record.currentTotalValidatorBalance;
583:         total += unstakeRequestsManager.balance();
584:         return total;
585:     }
```

Description

Yaodao : The function `totalControlled()` is used to calculate the total amount of ETH controlled by the protocol. And the result will be used in the main logic of the protocol, especially for the calculation of the interconversion of ETH and mETH.

```
    function totalControlled() public view returns (uint256) {
        OracleRecord memory record = oracle.latestRecord();
        uint256 total = 0;
        total += unallocatedETH;
        total += allocatedETHForDeposits;
        /// The total ETH deposited to the beacon chain must be decreased by the deposits processed
by the off-chain
        /// oracle since it will be accounted for in the currentTotalValidatorBalance from that poin
t onwards.
        total += totalDepositedInValidators - record.cumulativeProcessedDepositAmount;
        total += record.currentTotalValidatorBalance;
        total += unstakeRequestsManager.balance();
        return total;
    }
```

The formula is as follows: $$total = unallocatedETH + allocatedETHForDeposits + totalDepositedInValidators $$ $$- record.cumulativeProcessedDepositAmount + record.currentTotalValidatorBalance $$ $$+ unstakeRequestsManager.balance()$$

For `unallocatedETH`, it will be

1. increased the transferred amount in `stake()`.
2. decreased the amount `allocateToUnstakeRequestsManager` (added to `allocatedETHForDeposits`) and `allocateToDeposits` (transferred to `unstakeRequestsManager`) in `allocateETH()`.
3. increased the transferred amount from `unstakeRequestsManager` in `receiveFromUnstakeRequestsManager()`
4. increased the transferred amount from `TOP_UP_ROLE` in `topUp()`
5. increased the transferred amount from `ReturnsAggregator` in `receiveReturns()`

For `allocatedETHForDeposits`, it will be

1. increased the amount `allocateToDeposits` (deducted from the `unallocatedETH`) in `allocateETH()`
2. decreased the amount `amountDeposited` (added to `totalDepositedInValidators`) in `initiateValidatorsWithDeposits()`

For `totalDepositedInValidators`, it will be

1. increased the amount `amountDeposited` (deducted from the `allocatedETHForDeposits`) in `initiateValidatorsWithDeposits()`

For `record.cumulativeProcessedDepositAmount` and `record.currentTotalValidatorBalance`, these are controlled and updated by the `Oracle` contract. `record.cumulativeProcessedDepositAmount` is the total amount of ETH that has been deposited into and processed by the consensus layer, and the `record.currentTotalValidatorBalance` is the total amount of ETH in the consensus layer.

For `unstakeRequestsManager` contract, the contract does not accept external transfers, and sends the users' confirmed unstake amount and interacts with the Staking contract.

The `allocatedETHForDeposits` and `totalDepositedInValidators` are used to record the ETH amount used and are synchronized with `unallocatedETH`. The balance of `unstakeRequestsManager` contract is updated for the users' unstake and interacts with the Staking contract.

As a result, it needs to focus on is that the change of `unallocatedETH` and the correctness and validity of the data recorded in the `record` to judge whether the results of `totalControlled` are correct at all times.

For the above two variables, the following issues may exist and need to fix:

1. The function `receiveReturns()` is used to receive the rewards of consensus layer from the `ReturnReceiver` contracts, which is called in the `ReturnsAggregator.processReturns()`. As a result, that portion of the reward is not added into `totalControlled()` when the consensus layer reward has been claimed but `ReturnsAggregator.processReturns()` has not been called in time.
2. The function `initiateValidatorsWithDeposits()` is used to initiates new validators by sending ETH to the beacon chain deposit contract. After the call of `initiateValidatorsWithDeposits()`, the signature will be validated later by the consensus layer, and if found to be incorrect (for new validators) the deposit will fail, and the Ether will be lost. In this condition, the `allocatedETHForDeposits` and `totalDepositedInValidators`

are updated in the `initiateValidatorsWithDeposits()` but not recovered after the fail of deposit. As a result, the lost is not processed.

Recommendation

Yaodao : Recommend adding corresponding logic to fix these two conditions.

Client Response

Mitigated.Off-chain services verify all deposit signatures before staking

MNT-3:Missing Oracle Quorum Threshold Requirement and Dangerous Default Configuration

Category	Severity	Client Response	Contributor
Oracle Manipulation	**Medium**	**Declined**	**0x1337**

Code Reference

- code/src/OracleQuorumManager.sol#L224-L239
- code/src/OracleQuorumManager.sol#L119-L120
- code/src/OracleQuorumManager.sol#L128-L134

```
119:absoluteThreshold = 1;
120:        relativeThresholdBasisPoints = 0;

128:function _hasReachedQuroum(uint64 blockNumber, bytes32 recordHash) internal view returns (bool)
{
129:        uint256 numReports = recordHashCountByBlock[blockNumber][recordHash];
130:        uint256 numReporters = getRoleMemberCount(SERVICE_ORACLE_REPORTER);
131:
132:        return (numReports >= absoluteThreshold)
133:            && (numReports * _BASIS_POINTS_DENOMINATOR >= numReporters * relativeThresholdBasisP
oints);
134:    }

224:function setQuorumThresholds(uint16 absoluteThreshold_, uint16 relativeThresholdBasisPoints_)
225:        external
226:        onlyRole(QUORUM_MANAGER_ROLE)
227:    {
228:        if (relativeThresholdBasisPoints_ > _BASIS_POINTS_DENOMINATOR) {
229:            revert RelativeThresholdExceedsOne();
230:        }
231:
232:        emit ProtocolConfigChanged(
233:            this.setQuorumThresholds.selector,
234:            "setQuorumThresholds(uint16,uint16)",
235:            abi.encode(absoluteThreshold_, relativeThresholdBasisPoints_)
236:        );
237:        absoluteThreshold = absoluteThreshold_;
238:        relativeThresholdBasisPoints = relativeThresholdBasisPoints_;
239:    }
```

Description

0x1337 : In the `OracleQuorumManager` contract, both `absoluteThreshold` and `relativeThresholdBasisPoints` are used to determine if the oracle quorum threshold has been reached for a particular OracleRecord. While the Mantle Audit Guideline states "We are aware that the code initialises with a single oracle in the quorum set - but this is just for testing. For all issues you should assume that the mainnet configuration of at least 3 independent oracles will be used.", it does not reference the threshold requirement. For a threshold to be effective, there should be more stringent requirement than what's in the `setQuorumThresholds()` function, which in its current form, allows both thresholds to be set to 0, in which case a single malicious Oracle Record could percolate through the entire Mantle LSD system. The

default configuration in line 119-120 below is also unsafe, and allows a single malicious Oracle Record to be accepted by the `Oracle` contract.

```
        absoluteThreshold = 1;
        relativeThresholdBasisPoints = 0;
```

Recommendation

0x1337 : Similar to a properly configured multisig, the `OracleQuorumManager` contract should always ensure that a valid threshold is in place regardless of the total number of addresses that have the `SERVICE_ORACLE_REPORTER` role. Typically, this means a `relativeThresholdBasisPoints` of at least 5000 (50% of `_BASIS_POINTS_DENOMINATOR`), and an `absoluteThreshold` of 2 (when there are 3 SERVICE_ORACLE_REPORTER addresses, as the Mantle Audit Guideline indicates).

Client Response

Declined.The brief explicitly states that this configuration is for testing.
`Secure3 comment`: As stated in the audit guidance, "For all issues you should assume that the mainnet configuration of at least 3 independent oracles will be used. ". In the `receiveRecord` function, `numReports >= absoluteThreshold` will be checked. But in the `initialize` function, the `absoluteThreshold` will be 1. Relatively speaking, 2/3 is safer than 1/3. So the recommendation is that the `absoluteThreshold` value should be at least 2 when initialized to be safe.

MNT-4:Potential fail to pass `sanityCheckUpdate()` due to incorrect update the `minDepositPerValidator` and `maxDepositPerValidator`

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	Yaodao

Code Reference

- code/src/Oracle.sol#L436-L446

```
436:if (newDeposits < newValidators * minDepositPerValidator) {
437:                return (
438:                    "New deposits below min deposit per validator", newDeposits, newValidators * minDepositPerValidator
439:                );
440:            }
441:
442:            if (newDeposits > newValidators * maxDepositPerValidator) {
443:                return (
444:                    "New deposits above max deposit per validator", newDeposits, newValidators * maxDepositPerValidator
445:                );
446:            }
```

Description

Yaodao : The function `sanityCheckUpdate()` is used to check the incoming oracle update. According to the following codes, the deposits will be checked whether it is between `minDepositPerValidator` and `maxDepositPerValidator`.

```
if (newDeposits < newValidators * minDepositPerValidator) {
    return (
        "New deposits below min deposit per validator", newDeposits, newValidators * minDepositPerVa
lidator
    );
}

if (newDeposits > newValidators * maxDepositPerValidator) {
    return (
        "New deposits above max deposit per validator", newDeposits, newValidators * maxDepositPerVa
lidator
    );
}
```

However, the values of `minDepositPerValidator` and `maxDepositPerValidator` can be updated by the `ORACLE_MANAGER_ROLE`. As a result, although the deposit data is valid at the time of generation, it may fail to pass the `sanityCheckUpdate()` if `minDepositPerValidator` and `maxDepositPerValidator` are updated before call the `receiveRecord()`.

Recommendation

Yaodao : Recommend redesigning the update logic of `minDepositPerValidator` and `maxDepositPerValidator`.

Client Response

Acknowledged.Falls under 'assume competence' but this is non-obvious enough to warrant an acknowledgement. Changing of these fields must be done carefully

MNT-5: `minimumUnstakeBound_` need more limit

Category	Severity	Client Response	Contributor
Privilege Related	Low	Acknowledged	Satyam_

Code Reference

- code/src/Staking.sol#L647-L652

```
647:function setMinimumUnstakeBound(uint256 minimumUnstakeBound_) external onlyRole(STAKING_MANAGER_
ROLE) {
648:        minimumUnstakeBound = minimumUnstakeBound_;
649:        emit ProtocolConfigChanged(
650:            this.setMinimumUnstakeBound.selector, "setMinimumUnstakeBound(uint256)", abi.encode
(minimumUnstakeBound_)
651:        );
652:    }
```

Description

Satyam_ : Staking.setMinimumUnstakeBound is used by the staking manager through which he can edit `minimumUnstakeBound_` that can be greater than the previous `minimumUnstakeBound_` limit , which will make the user to unstake their bound.

```
 /// @notice Sets the minimum amount of mETH users can unstake.
    function setMinimumUnstakeBound(uint256 minimumUnstakeBound_) external onlyRole(STAKING_MANAGER_
ROLE) {
        minimumUnstakeBound = minimumUnstakeBound_;//@audit manager can set minimumUnstakeBound to h
igh value
        emit ProtocolConfigChanged(
            this.setMinimumUnstakeBound.selector, "setMinimumUnstakeBound(uint256)", abi.encode(mini
mumUnstakeBound_)
        );
    }
```

- suppose user stake their amount in the staking contract, when it has a `minimumUnstakeBound = 0.01 ether;` and after a month later staking_manager edit the `minimumUnstakeBound` and set it to any arbitrary high value or suppose stake manager set minimumUnstakeBound value to greater than or equal to 1eth, now user who stake their amount lesser than `1eth`, at the time when `minimumUnstakeBound = 0.01 ether;` are unable to unstake their bound amount, which makes the user fund stuck in the contract.

Recommendation

Satyam_ : make a minimum limit , so that a stake manager unable to set `minimumUnstakeBound` to any arbitrary high number , introduced a require condition , where new minimumUnstakeBound should lesser than that value.

Client Response

Acknowledged.Goes against guidance document but we agree that some bounds here would be sensible.

MNT-6: `min` variable should be less than `max` variable

Category	Severity	Client Response	Contributor
Logical	Low	Declined	Yaodao, BradMoonUESTC

Code Reference

- code/src/Staking.sol#L470-L476
- code/src/Staking.sol#L674-L679
- code/src/Staking.sol#L683-L688
- code/src/Oracle.sol#L639-L776

```
470:if (validator.depositAmount < minimumDepositAmount) {
471:                revert MinimumValidatorDepositNotSatisfied();
472:            }
473:
474:            if (validator.depositAmount > maximumDepositAmount) {
475:                revert MaximumValidatorDepositExceeded();
476:            }

639:function setFinalizationBlockNumberDelta(uint256 finalizationBlockNumberDelta_)
640:        external
641:        onlyRole(ORACLE_MANAGER_ROLE)
642:    {
643:        if (
644:            finalizationBlockNumberDelta_ == 0
645:                || finalizationBlockNumberDelta_ > _FINALIZATION_BLOCK_NUMBER_DELTA_UPPER_BOUND
646:        ) {
647:            revert InvalidConfiguration();
648:        }
649:
650:        finalizationBlockNumberDelta = finalizationBlockNumberDelta_;
651:        emit ProtocolConfigChanged(
652:            this.setFinalizationBlockNumberDelta.selector,
653:            "setFinalizationBlockNumberDelta(uint256)",
654:            abi.encode(finalizationBlockNumberDelta_)
655:        );
656:    }
657:
658:    /// @inheritdoc IOracleManager
659:    /// @dev See also {oracleUpdater}.
660:    function setOracleUpdater(address newUpdater) external onlyRole(ORACLE_MANAGER_ROLE) notZero
Address(newUpdater) {
661:        oracleUpdater = newUpdater;
662:        emit ProtocolConfigChanged(this.setOracleUpdater.selector, "setOracleUpdater(address)",
abi.encode(newUpdater));
663:    }
664:
665:    /// @notice Sets min deposit per validator in the contract.
666:    /// See also {minDepositPerValidator}.
667:    /// @param minDepositPerValidator_ The new min deposit per validator.
668:    function setMinDepositPerValidator(uint256 minDepositPerValidator_) external onlyRole(ORACLE
_MANAGER_ROLE) {
```

```
669:         minDepositPerValidator = minDepositPerValidator_;
670:         emit ProtocolConfigChanged(
671:             this.setMinDepositPerValidator.selector,
672:             "setMinDepositPerValidator(uint256)",
673:             abi.encode(minDepositPerValidator_)
674:         );
675:     }
676:
677:     /// @notice Sets max deposit per validator in the contract.
678:     /// See also {maxDepositPerValidator}.
679:     /// @param maxDepositPerValidator_ The new max deposit per validator.
680:     function setMaxDepositPerValidator(uint256 maxDepositPerValidator_) external onlyRole(ORACLE
_MANAGER_ROLE) {
681:         maxDepositPerValidator = maxDepositPerValidator_;
682:         emit ProtocolConfigChanged(
683:             this.setMaxDepositPerValidator.selector,
684:             "setMaxDepositPerValidator(uint256)",
685:             abi.encode(maxDepositPerValidator)
686:         );
687:     }
688:
689:     /// @notice Sets min consensus layer balance per validator in the contract.
690:     /// See also {minConsensusLayerBalancePerValidator}.
691:     /// @param minConsensusLayerBalancePerValidator_ The new min consensus layer balance per val
idator.
692:     function setMinConsensusLayerBalancePerValidator(uint256 minConsensusLayerBalancePerValidato
r_)
693:         external
694:         onlyRole(ORACLE_MANAGER_ROLE)
695:     {
696:         minConsensusLayerBalancePerValidator = minConsensusLayerBalancePerValidator_;
697:         emit ProtocolConfigChanged(
698:             this.setMinConsensusLayerBalancePerValidator.selector,
699:             "setMinConsensusLayerBalancePerValidator(uint256)",
700:             abi.encode(minConsensusLayerBalancePerValidator_)
701:         );
702:     }
703:
704:     /// @notice Sets max consensus layer balance per validator in the contract.
705:     /// See also {maxConsensusLayerBalancePerValidator}.
706:     /// @param maxConsensusLayerBalancePerValidator_ The new max consensus layer balance per val
idator.
707:     function setMaxConsensusLayerBalancePerValidator(uint256 maxConsensusLayerBalance-
```

```
PerValidator_)
        external
        onlyRole(ORACLE_MANAGER_ROLE)
    {
        maxConsensusLayerBalancePerValidator = maxConsensusLayerBalancePerValidator_;
        emit ProtocolConfigChanged(
            this.setMaxConsensusLayerBalancePerValidator.selector,
            "setMaxConsensusLayerBalancePerValidator(uint256)",
            abi.encode(maxConsensusLayerBalancePerValidator_)
        );
    }

    /// @notice Sets min consensus layer gain per block in the contract.
    /// See also {minConsensusLayerGainPerBlockPPT}.
    /// @param minConsensusLayerGainPerBlockPPT_ The new min consensus layer gain per block in p
arts per trillion.
    function setMinConsensusLayerGainPerBlockPPT(uint40 minConsensusLayerGainPerBlockPPT_)
        external
        onlyRole(ORACLE_MANAGER_ROLE)
        onlyFractionLeqOne(minConsensusLayerGainPerBlockPPT_, _PPT_DENOMINATOR)
    {
        minConsensusLayerGainPerBlockPPT = minConsensusLayerGainPerBlockPPT_;
        emit ProtocolConfigChanged(
            this.setMinConsensusLayerGainPerBlockPPT.selector,
            "setMinConsensusLayerGainPerBlockPPT(uint40)",
            abi.encode(minConsensusLayerGainPerBlockPPT_)
        );
    }

    /// @notice Sets max consensus layer gain per block in the contract.
    /// See also {maxConsensusLayerGainPerBlockPPT}.
    /// @param maxConsensusLayerGainPerBlockPPT_ The new max consensus layer gain per block in p
arts per million.
    function setMaxConsensusLayerGainPerBlockPPT(uint40 maxConsensusLayerGainPerBlockPPT_)
        external
        onlyRole(ORACLE_MANAGER_ROLE)
        onlyFractionLeqOne(maxConsensusLayerGainPerBlockPPT_, _PPT_DENOMINATOR)
    {
        maxConsensusLayerGainPerBlockPPT = maxConsensusLayerGainPerBlockPPT_;
        emit ProtocolConfigChanged(
            this.setMaxConsensusLayerGainPerBlockPPT.selector,
```

```
746:             "setMaxConsensusLayerGainPerBlockPPT(uint40)",
747:             abi.encode(maxConsensusLayerGainPerBlockPPT_)
748:         );
749:     }
750:
751:     /// @notice Sets max consensus layer loss per block in the contract.
752:     /// See also {maxConsensusLayerLossPPM}.
753:     /// @param maxConsensusLayerLossPPM_ The new max consensus layer loss per block in parts per
million.
754:     function setMaxConsensusLayerLossPPM(uint24 maxConsensusLayerLossPPM_)
755:         external
756:         onlyRole(ORACLE_MANAGER_ROLE)
757:         onlyFractionLeqOne(maxConsensusLayerLossPPM_, _PPM_DENOMINATOR)
758:     {
759:         maxConsensusLayerLossPPM = maxConsensusLayerLossPPM_;
760:         emit ProtocolConfigChanged(
761:             this.setMaxConsensusLayerLossPPM.selector,
762:             "setMaxConsensusLayerLossPPM(uint24)",
763:             abi.encode(maxConsensusLayerLossPPM_)
764:         );
765:     }
766:
767:     /// @notice Sets the minimum report size.
768:     /// See also {minReportSizeBlocks}.
769:     /// @param minReportSizeBlocks_ The new minimum report size, in blocks.
770:     function setMinReportSizeBlocks(uint16 minReportSizeBlocks_) external onlyRole(ORACLE_MANAGE
R_ROLE) {
771:         // Sanity check on upper bound is covered by uint16 which is ~9 days.
772:         minReportSizeBlocks = minReportSizeBlocks_;
773:         emit ProtocolConfigChanged(
774:             this.setMinReportSizeBlocks.selector, "setMinReportSizeBlocks(uint16)", abi.encode(m
inReportSizeBlocks_)
775:         );
776:     }

674:function setMinimumDepositAmount(uint256 minimumDepositAmount_) external onlyRole(STAKING_MANAGE
R_ROLE) {
675:         minimumDepositAmount = minimumDepositAmount_;
676:         emit ProtocolConfigChanged(
677:             this.setMinimumDepositAmount.selector, "setMinimumDepositAmount(uint256)", abi.encod
e(minimumDepositAmount_)
678:         );
679:     }
```

```
683:function setMaximumDepositAmount(uint256 maximumDepositAmount_) external onlyRole(STAKING_MANAGE
R_ROLE) {
684:        maximumDepositAmount = maximumDepositAmount_;
685:        emit ProtocolConfigChanged(
686:            this.setMaximumDepositAmount.selector, "setMaximumDepositAmount(uint256)", abi.encod
e(maximumDepositAmount_)
687:        );
688:    }
```

Description

Yaodao : The function `initiateValidatorsWithDeposits()` is used to initiate the validators and will check the `validator.depositAmount` between `minimumDepositAmount` and `maximumDepositAmount`. The initialized values of `minimumDepositAmount` and `maximumDepositAmount` are both 32 ether, so the `validator.depositAmount` must be 32 ether.

However, the values of `minimumDepositAmount` and `maximumDepositAmount` can be updated by the functions `setMinimumDepositAmount()` and `setMaximumDepositAmount()` without limit checks.

As a result, the value of `minimumDepositAmount` may be over `maximumDepositAmount` and the check in the `initiateValidatorsWithDeposits()` will always able unable to pass.

The similar variables are `Oracle.minDepositPerValidator` and `Oracle.maxDepositPerValidator`, `Oracle.minConsensusLayerGainPerBlockPPT` and `Oracle.maxConsensusLayerGainPerBlockPPT`, `Oracle.minConsensusLayerBalancePerValidator` and `Oracle.maxConsensusLayerBalancePerValidator`.

BradMoonUESTC : A logical flaw exists within the functions `setMinDepositPerValidator`, `setMaxDepositPerValidator`, `setMinConsensusLayerBalancePerValidator`, and `setMaxConsensusLayerBalancePerValidator`. While both pairs of these functions allow the administrator to set maximum and minimum values for deposits or balances, they lack safeguards to ensure that the minimum does not exceed the maximum. This oversight creates scenarios where it's impossible to make valid deposits or set effective balance ranges, as the available range (i.e., deposits between the minimum and maximum values) becomes nonexistent.

Recommendation

Yaodao : Recommend adding check to ensure the value of `minimumDepositAmount` will never over `maximumDepositAmount`.

BradMoonUESTC : To rectify this flaw, introduce logic checks in the `setMinDepositPerValidator`, `setMaxDepositPerValidator`, `setMinConsensusLayerBalancePerValidator`, and `setMaxConsensusLayerBalancePerValidator` functions to guarantee that the set minimum value is always less than or equal to the maximum value.

For example, in the `setMaxDepositPerValidator` function, implement the following check:

```
if (maxDepositPerValidator_ < minDepositPerValidator) {
    revert InvalidConfiguration();
}
```

Similarly, within the `setMinDepositPerValidator` function, incorporate the check:

```
if (minDepositPerValidator_ > maxDepositPerValidator) {
    revert InvalidConfiguration();
}
```

By incorporating these logical checks, you ensure the presence of a valid deposit range whenever setting new minimum or maximum values.

Client Response

Declined.Our team will maintain the configuration correct.
`Secure3 comment`:We have confidence in the team's management, meanwhile we also believe that configuring the code correctly is safer. Because the min and max variables are set separately, this will make the relationship between the two not so close.Wrong settings can cause Dos.

MNT-7:Missing Zero Address Check in `Staking::initialize` function

Category	Severity	Client Response	Contributor
Code Style	Low	Declined	Xi_Zi

Code Reference

- code/src/Staking.sol#L260-L293

```
260:function initialize(Init memory init) external initializer {
261:        __AccessControlEnumerable_init();
262:
263:        _grantRole(DEFAULT_ADMIN_ROLE, init.admin);
264:        _grantRole(STAKING_MANAGER_ROLE, init.manager);
265:        _grantRole(ALLOCATOR_SERVICE_ROLE, init.allocatorService);
266:        _grantRole(INITIATOR_SERVICE_ROLE, init.initiatorService);
267:        // Intentionally does not set anyone as the TOP_UP_ROLE as it will only be granted
268:        // in the off-chance that the top up functionality is required.
269:
270:        // Set up roles for the staking allowlist. Intentionally do not grant anyone the
271:        // STAKING_ALLOWLIST_MANAGER_ROLE as it will only be granted later.
272:        _setRoleAdmin(STAKING_ALLOWLIST_MANAGER_ROLE, STAKING_MANAGER_ROLE);
273:        _setRoleAdmin(STAKING_ALLOWLIST_ROLE, STAKING_ALLOWLIST_MANAGER_ROLE);
274:
275:        mETH = init.mETH;
276:        depositContract = init.depositContract;
277:        oracle = init.oracle;
278:        pauser = init.pauser;
279:        returnsAggregator = init.returnsAggregator;
280:        unstakeRequestsManager = init.unstakeRequestsManager;
281:        withdrawalWallet = init.withdrawalWallet;
282:
283:        minimumStakeBound = 0.1 ether;
284:        minimumUnstakeBound = 0.01 ether;
285:        minimumDepositAmount = 32 ether;
286:        maximumDepositAmount = 32 ether;
287:        isStakingAllowlist = true;
288:        initializationBlockNumber = block.number;
289:
290:        // Set the maximum mETH supply to some sensible amount which is expected to be changed a
s the
291:        // protocol ramps up.
292:        maximumMETHSupply = 1024 ether;
293:    }
```

Description

Xi_Zi : In the provided staking contract, the setWithdrawalWallet function includes a check to ensure that the withdrawalWallet_ address is not a zero address. However, the initialize function, which also sets the withdrawalWallet,

lacks this zero address check. This inconsistency can lead to potential issues if the initialize function is inadvertently called with a zero address for the withdrawalWallet.

```
    function initialize(Init memory init) external initializer {
        . . .

        withdrawalWallet = init.withdrawalWallet;
         . . .

    }

    function setWithdrawalWallet(address withdrawalWallet_)
        external
        onlyRole(STAKING_MANAGER_ROLE)
        notZeroAddress(withdrawalWallet_)
    {
        withdrawalWallet = withdrawalWallet_;
        emit ProtocolConfigChanged(
            this.setWithdrawalWallet.selector, "setWithdrawalWallet(address)", abi.encode(withdrawal
Wallet_)
        );
    }
```

Recommendation

Xi_Zi : To maintain consistency and ensure the security of the contract, it's recommended to add a zero address check in the initialize function, similar to the check in the setWithdrawalWallet function.

Client Response

Declined.Our team will maintain the configuration correct.
`Secure3 comment`:We have confidence in the team's management, but in the spirit of decentralization, we believe that configuring the code correctly is safer.The parameter that are used in init() function to initialize the state variable,these state variable are used in other function to perform operation. since it lacks zero address validation, it will be problematic if there is error in these state variable. some of the function will loss their functionality which can cause the redeployment of contract.

MNT-8:feesBasisPoints need more limit

Category	Severity	Client Response	Contributor
Privilege Related	Low	Declined	Xi_Zi

Code Reference

- code/src/ReturnsAggregator.sol#L167-L177

```
167:if (newBasisPoints > _BASIS_POINTS_DENOMINATOR) {
168:            revert InvalidConfiguration();
169:        }
170:
171:        feesBasisPoints = newBasisPoints;
172:        emit ProtocolConfigChanged(
173:            this.setFeeBasisPoints.selector, "setFeeBasisPoints(uint16)", abi.encode(newBasisPoi
nts)
174:        );
175:    }
176:
177:    receive() external payable {}
```

Description

Xi_Zi : In the setFeeBasisPoints function, there is a validation for feesBasisPoints to ensure it does not exceed _BASIS_POINTS_DENOMINATOR. However, the function does not set an upper limit for feesBasisPoints. Given the centralized nature of this function, if feesBasisPoints is set to an excessively large value, it could result in the project imposing exorbitantly high fees. This could unfairly burden users and potentially harm the project's reputation.

```
    function setFeeBasisPoints(uint16 newBasisPoints) external onlyRole(AGGREGATOR_MANAGER_ROLE) {
        if (newBasisPoints > _BASIS_POINTS_DENOMINATOR) {
            revert InvalidConfiguration();
        }

        feesBasisPoints = newBasisPoints;
        emit ProtocolConfigChanged(
            this.setFeeBasisPoints.selector, "setFeeBasisPoints(uint16)", abi.encode(newBasisPoints)
        );
    }
```

Recommendation

Xi_Zi : It is recommended to introduce an upper limit for feesBasisPoints within the setFeeBasisPoints function. For instance, you could set its maximum value to 2_000 (representing 20%), ensuring fees remain within a reasonable range. This would prevent potential misuse.

Client Response

Declined.Subjective, our team will maintain the configuration correct.
`Secure3 comment`: We have confidence in the team's management, meanwhile we also believe that configuring the code correctly is safe. It is reasonable to charge fees, but this base cannot be infinite and must be limited to a reasonable range, otherwise users may suffer huge losses without knowing it.

MNT-9:Unlocked pragma

Category	Severity	Client Response	Contributor
Language Specific	Informational	Acknowledged	rajatbeladiya

Code Reference

- code/src/Oracle.sol#L2
- code/src/OracleQuorumManager.sol#L2
- code/src/Pauser.sol#L2
- code/src/ReturnsAggregator.sol#L2
- code/src/Staking.sol#L2

```
2:pragma solidity ^0.8.20;

2:pragma solidity ^0.8.20;

2:pragma solidity ^0.8.20;

2:pragma solidity ^0.8.20;

2:pragma solidity ^0.8.20;
```

Description

rajatbeladiya : Mantle LSD has files with pragma solidity version number with ^0.8.20. The caret (^) points to unlocked pragma, meaning compiler will use the specified version or above. It's good practice to use specific solidity version to know compiler bug fixes and optimisations were enabled at the time of compiling the contract.

Recommendation

rajatbeladiya : Use specific solidity version

```
pragma solidity 0.8.20;
```

Client Response

Acknowledged.

MNT-10:Missing Event Setter in `Staking::reclaimAllocatedETHSurplus` function

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	Xi_Zi

Code Reference

- code/src/Staking.sol#L404-L408

```
404:function reclaimAllocatedETHSurplus() external onlyRole(STAKING_MANAGER_ROLE) {
405:        // Calls the receiveFromUnstakeRequestsManager() where we perform
406:        // the accounting.
407:        unstakeRequestsManager.withdrawAllocatedETHSurplus();
408:    }
```

Description

Xi_Zi : The staking contract contains critical operations that lack event logging.

```
    function reclaimAllocatedETHSurplus() external onlyRole(STAKING_MANAGER_ROLE) {
        // Calls the receiveFromUnstakeRequestsManager() where we perform
        // the accounting.
        unstakeRequestsManager.withdrawAllocatedETHSurplus();
    }

    function withdrawAllocatedETHSurplus() external onlyStakingContract {
        uint256 toSend = allocatedETHSurplus();
        if (toSend == 0) {
            return;
        }
        allocatedETHForClaims -= toSend;
        stakingContract.receiveFromUnstakeRequestsManager{value: toSend}();
    }

    function receiveFromUnstakeRequestsManager() external payable onlyUnstakeRequestsManager {
        unallocatedETH += msg.value;
    }
```

Recommendation

Xi_Zi : it's recommended to add relevant events to log important information

Client Response

Acknowledged.An event isn't needed but it wouldn't hurt to include one.

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 Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. 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 Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. 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.