Competitive Security Assessment

Lagrange_Update

Apr 10th, 2024

Secure3

secure3.io

Summary 3

Overview 4

Audit Scope 5

Code Assessment Findings 6

LGR-1 `addBlsPubKeys` does not check `BlsPubKeys` for duplicates 8

LGR-2 The quorumNumbers may contain duplicate members 11

LGR-3 Ownership change should use two-step process 13

LGR-4 Incorrect if condition in `updateQuorumMultiplier` prevents adding new multipliers to the quorum. 14

LGR-5 Duplicates can increase the voteweight in `weightOfOperator` 17

LGR-6 missing indexed keyword in event log 22

LGR-7 Using `calldata` instead of `memory` for read-only arguments in external functions saves gas. 23

Disclaimer 25

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.

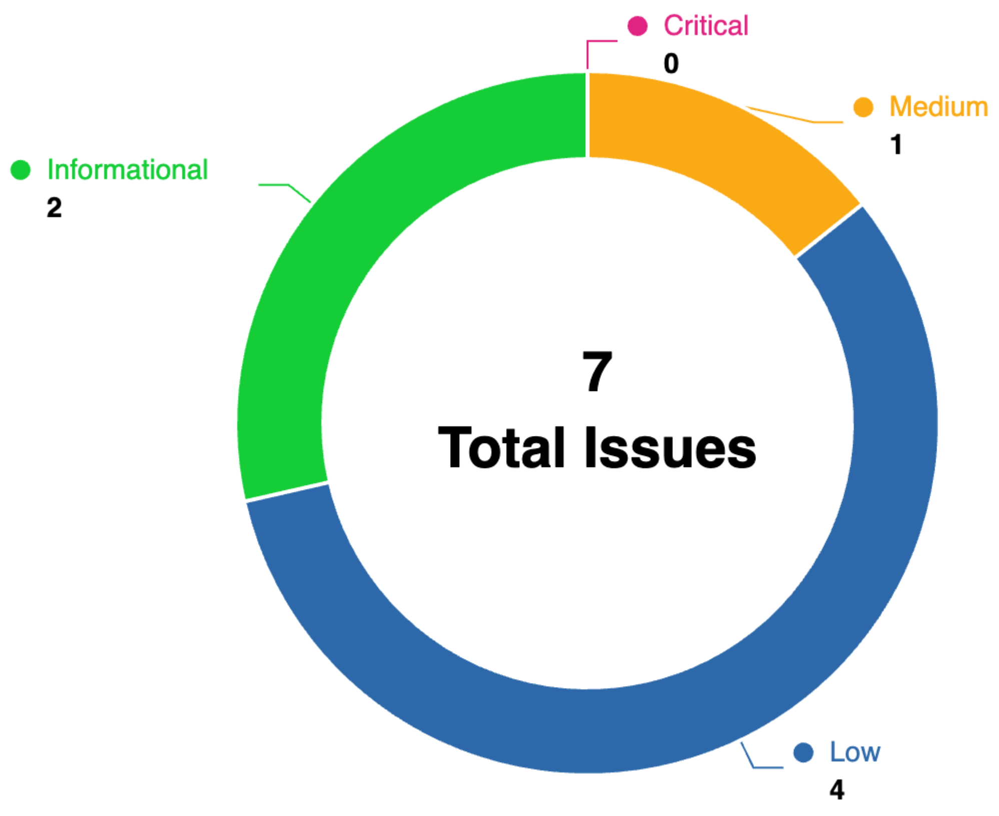
Overview

Project Name	Lagrange_Update
Language	Solidity
Codebase	• https://github.com/Lagrange-Labs/lagrange-contracts-audit/ • audit version - 65a446f0bc29ce8080582294d710043dc2a1a9aa • final version - 3e68e8e4cea8f1f3e271ac6743f03b86e1a0cead
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
src/protocol/LagrangeCommittee.sol	a570cbf2aef1cb30ea5030417d5d257c4100231d9e01a150aa85676cd0015957
src/protocol/VoteWeigher.sol	61d20fcd2a090e655cac7ae550a9113812ccc3e5b6b84be5ff892e149a28d41a
src/protocol/LagrangeService.sol	cdceb2bc119e6a527fa47d3bba2972327c2930af940aac8bb5a8f9a02541617d
src/interfaces/ILagrangeCommittee.sol	2bbd2b60b0611ea8d1d0909d3094fca7cc3871deda8c655cd8cdabca49db2a4b
src/library/StakeManager.sol	17eb34a84caf80a1dca66e21eac9b6ea63f9a3cfb9921e2a5729861cc488244a
src/library/EigenAdapter.sol	56c61f64fd038a3c0180830255d42291c2baf2def38a4c4f9814b4d655c428ac
src/interfaces/ILagrangeService.sol	c57c854c224b585db4963dbad10a82e61e8b6b0b505acb5b95c09f90efe5f0fe
src/interfaces/IVoteWeigher.sol	451798d02b6b51b5ffafd2ce08e0b923578b40d8c7825ae3a8bc9f433b194eed
src/interfaces/IStakeManager.sol	167863543303e1370f5d63efa6e5a4bd0c461dae2d771973439fb0b69bd5eadf

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
LGR-1	`addBlsPubKeys` does not check `BlsPubKeys` for duplicates	Logical	Medium	Fixed	biakia, crjr0629
LGR-2	The quorumNumbers may contain duplicate members	Logical	Low	Fixed	1nc0gn170, biakia
LGR-3	Ownership change should use two-step process	Privilege Related	Low	Acknowledged	biakia
LGR-4	Incorrect if condition in `updateQuorumMultiplier` prevents adding new multipliers to the quorum.	Logical	Low	Fixed	1nc0gn170, biakia
LGR-5	Duplicates can increase the voteweight in `weightOfOperator`	Logical	Low	Fixed	1nc0gn170
LGR-6	missing indexed keyword in event log	Code Style	Informational	Fixed	newway55

LGR-7	Using `calldata` instead of `memory` for read-only arguments in external functions saves gas.	Gas Optimization	Informational	Fixed	n16h7m4r3

LGR-1: `addBlsPubKeys` does not check `BlsPubKeys` for duplicates

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	biakia, crjr0629

Code Reference

- code/src/protocol/LagrangeCommittee.sol#L59-L70
- code/src/protocol/LagrangeCommittee.sol#L67-L70
- code/src/protocol/LagrangeCommittee.sol#L375-L382

```
59: // Adds address stake data and flags it for committee addition
60:     function addOperator(address operator, address signAddress, uint256[2][] memory blsPubKeys) public onlyService {
61:         _validateBlsPubKeys(blsPubKeys);
62:         _registerOperator(operator, signAddress, blsPubKeys);
63:     }
64:
65:     // Adds address stake data and flags it for committee addition
66:     function addBlsPubKeys(address operator, uint256[2][] memory additionalBlsPubKeys) public onlyService {
67:         _validateBlsPubKeys(additionalBlsPubKeys);
68:         _addBlsPubKeys(operator, additionalBlsPubKeys);
69:     }
```

```
67: function addBlsPubKeys(address operator, uint256[2][] memory additionalBlsPubKeys) public onlyService {
68:         _validateBlsPubKeys(additionalBlsPubKeys);
69:         _addBlsPubKeys(operator, additionalBlsPubKeys);
70:     }
```

```
375: function _addBlsPubKeys(address _operator, uint256[2][] memory _additionalBlsPubKeys) internal {
376:         OperatorStatus storage _opStatus = operatorsStatus[_operator];
377:         require(_opStatus.blsPubKeys.length != 0, "Operator is not registered.");
378:         uint256 _length = _additionalBlsPubKeys.length;
379:         for (uint256 i; i < _length; i++) {
380:             _opStatus.blsPubKeys.push(_additionalBlsPubKeys[i]);
381:         }
382:     }
```

Description

biakia: The function `addBlsPubKeys` will add `BlsPubKeys` to the `OperatorStatus` but lacks of checking the `BlsPubKeys` for duplicates:

```
function _addBlsPubKeys(address _operator, uint256[2][] memory _additionalBlsPubKeys) internal {
        OperatorStatus storage _opStatus = operatorsStatus[_operator];
        require(_opStatus.blsPubKeys.length != 0, "Operator is not registered.");
        uint256 _length = _additionalBlsPubKeys.length;
        for (uint256 i; i < _length; i++) {
            _opStatus.blsPubKeys.push(_additionalBlsPubKeys[i]);
        }
    }
```

So the `` `_opStatus.blsPubKeys` `` may contain duplicated members.
In function `` `update` ``, the `` `_opStatus.blsPubKeys` `` is used to compute the committee leaf:

```
unchecked {
                    for (uint256 j; _remained > 0;) {
                        uint96 _individualVotingPower;
                        if (_remained >= _maxWeight + _minWeight) {
                            _individualVotingPower = _maxWeight;
                        } else if (_remained > _maxWeight) {
                            _individualVotingPower = _minWeight;
                        } else {
                            _individualVotingPower = _remained;
                        }
                        _remained -= _individualVotingPower;
                        _committeeLeaves[_leafCounter] =
                            _leafHash(_operator, _opStatus.blsPubKeys[j], _individualVotingPower);
                        j++;
                        _leafCounter++;
                    }
                }
```

If a `` `_operator` `` has duplicated `` `blsPubKeys` ``, then it is likely to compute the same committee leaf. These committee leaves are used later to compute the merkle tree root. If the merkle tree has same committee leaves, then it is likely to be unsafe.
crjr0629: In this version, an operator can have mutiple public keys, there is no check on whether a `` `blspubkey` `` is already used by an operator. In one single operator there will be a miscalculation of the `` `_checkVotingPower` `` function as the array will have duplicate values. on function `` `update()` `` the `` `leafHash` `` on each step will be the same if `` `_indivualVotingPower` `` and `` `blsPubKeys` `` of an operator are the same. This can lead to unexpected behavior or erroneous data.
Likewise, there is no function to remove blspubkeys from operators.

Recommendation

biakia: Consider adding a check in `` `_addBlsPubKeys` ``:

```
function _addBlsPubKeys(address _operator, uint256[2][] memory _additionalBlsPubKeys) internal {
        OperatorStatus storage _opStatus = operatorsStatus[_operator];
        require(_opStatus.blsPubKeys.length != 0, "Operator is not registered.");
        uint256 _length = _additionalBlsPubKeys.length;
        for (uint256 i; i < _length; i++) {
            for(uint256 j;j<_opStatus.blsPubKeys.length;j++){
                if(_opStatus.blsPubKeys[j][0] == _additionalBlsPubKeys[i][0]
                   && _opStatus.blsPubKeys[j][1] == _additionalBlsPubKeys[i][1]){
                    revert DuplicatedBlsPubkeys;
                   }
            }
            _opStatus.blsPubKeys.push(_additionalBlsPubKeys[i]);
        }
    }
```

crjr0629: - add a check to see if the public key is already used by an operator.

- add a function to remove / modify pubkey entries from an operator.

Client Response

client response for biakia: Fixed -
client response for crjr0629: Fixed -

LGR-2:The quorumNumbers may contain duplicate members

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	1nc0gn170, biakia

Code Reference

- code/src/protocol/VoteWeigher.sol#L38
- code/src/protocol/VoteWeigher.sol#L38-L45

```
38: function addQuorumMultiplier(uint8 quorumNumber, TokenMultiplier[] memory multipliers) external
onlyOwner {
```

```
38: function addQuorumMultiplier(uint8 quorumNumber, TokenMultiplier[] memory multipliers) external
onlyOwner {
39:         require(quorumMultipliers[quorumNumber].length == 0, "Quorum already exists");
40:         for (uint256 i; i < multipliers.length; i++) {
41:             quorumMultipliers[quorumNumber].push(multipliers[i]);
42:         }
43:         quorumNumbers.push(quorumNumber);
44:         emit QuorumAdded(quorumNumber, multipliers);
45:     }
```

Description

1nc0gn170: TESTTEST
biakia: The function `addQuorumMultiplier` lacks of check whether the `multipliers` is empty:

```
function addQuorumMultiplier(uint8 quorumNumber, TokenMultiplier[] memory multipliers) external on
lyOwner {
        require(quorumMultipliers[quorumNumber].length == 0, "Quorum already exists");
        for (uint256 i; i < multipliers.length; i++) {
            quorumMultipliers[quorumNumber].push(multipliers[i]);
        }
        quorumNumbers.push(quorumNumber);
        emit QuorumAdded(quorumNumber, multipliers);
    }
```

When the `multipliers` is empty, the for loop will be skipped and the `quorumNumber` will be directly pushed into `quorumNumbers`. In this case, the `quorumMultipliers[quorumNumber].length` is still 0. The owner can call `addQuorumMultiplier` again to add the same `quorumNumber` into `quorumNumbers`, which means the `quorumNumbers` may contain duplicate members.

Recommendation

1nc0gn170:

```
    function addQuorumMultiplier(uint8 quorumNumber, TokenMultiplier[] memory multipliers) externa
l onlyOwner {
+        require(multipliers.length > 0, "Empty Multiplers");
        require(quorumMultipliers[quorumNumber].length == 0, "Quorum already exists");
        for (uint256 i; i < multipliers.length; i++) {
            quorumMultipliers[quorumNumber].push(multipliers[i]);
        }
        quorumNumbers.push(quorumNumber);
        emit QuorumAdded(quorumNumber, multipliers);
    }
```

biakia: Consider adding a check:

```
function addQuorumMultiplier(uint8 quorumNumber, TokenMultiplier[] memory multipliers) external on
lyOwner {
        require(multipliers.length>0,"invalid multipliers");
        require(quorumMultipliers[quorumNumber].length == 0, "Quorum already exists");
        for (uint256 i; i < multipliers.length; i++) {
            quorumMultipliers[quorumNumber].push(multipliers[i]);
        }
        quorumNumbers.push(quorumNumber);
        emit QuorumAdded(quorumNumber, multipliers);
    }
```

Client Response

client response for 1nc0gn170: Fixed - fix by this
https://github.com/Lagrange-Labs/lagrange-contracts-audit/commit/3e68e8e4cea8f1f3e271ac6743f03b86e1a0cead
client response for biakia: Fixed - fix by this
https://github.com/Lagrange-Labs/lagrange-contracts-audit/commit/3e68e8e4cea8f1f3e271ac6743f03b86e1a0cead

LGR-3:Ownership change should use two-step process

Category	Severity	Client Response	Contributor
Privilege Related	Low	Acknowledged	biakia

Code Reference

- code/src/protocol/LagrangeCommittee.sol#L11

```
11: contract LagrangeCommittee is Initializable, OwnableUpgradeable, ILagrangeCommittee {
```

- code/src/protocol/LagrangeService.sol#L14

```
14: contract LagrangeService is Initializable, OwnableUpgradeable, ILagrangeService {
```

- code/src/protocol/VoteWeigher.sol#L14

```
14: contract VoteWeigher is Initializable, OwnableUpgradeable, IVoteWeigher {
```

Description

biakia: The contract `VoteWeigher`, `LagrangeService` and `LagrangeCommittee` do not implement a two-step process for transferring ownership, so ownership of the contract can be easily lost when making a mistake when transferring ownership.

Recommendation

biakia: Consider Ownable2StepUpgradeable(https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/Ownable2StepUpgradeable.sol) instead.

Client Response

client response for biakia: Acknowledged
Secure3: Acknowledged

LGR-4:Incorrect if condition in `updateQuorumMultiplier` prevents adding new multipliers to the quorum.

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	1nc0gn170, biakia

Code Reference

- code/src/protocol/VoteWeigher.sol#L61-L69
- code/src/protocol/VoteWeigher.sol#L62-L65

```
61: function updateQuorumMultiplier(uint8 quorumNumber, uint256 index, TokenMultiplier memory multiplier) external onlyOwner {
62:         require(quorumMultipliers[quorumNumber].length > index, "Index out of bounds");
63:         if (quorumMultipliers[quorumNumber].length == index) {
64:             quorumMultipliers[quorumNumber].push(multiplier);
65:         } else {
66:             quorumMultipliers[quorumNumber][index] = multiplier;
67:         }
68:         emit QuorumUpdated(quorumNumber, index, multiplier);
69:     }
```

```
62: require(quorumMultipliers[quorumNumber].length > index, "Index out of bounds");
63:         if (quorumMultipliers[quorumNumber].length == index) {
64:             quorumMultipliers[quorumNumber].push(multiplier);
65:         } else {
```

Description

1nc0gn170: The function `` `VoteWeigher.updateQuorumMultiplier` `` is intended to

- update existing multiplier of a particular quorm at a given index.
- adding a new multiplier to the existing multiplier. (This is to avoid removing and adding the whole quorum data again in case whe a new multiplier needs to be added).

Due to an incorrect validation, the latter one is not possible.

```
    function updateQuorumMultiplier(uint8 quorumNumber, uint256 index, TokenMultiplier memory multiplier) external onlyOwner {
        require(quorumMultipliers[quorumNumber].length > index, "Index out of bounds");
        if (quorumMultipliers[quorumNumber].length == index) {
            quorumMultipliers[quorumNumber].push(multiplier);
        }
        else { // Update existing }
```

This the function function checks that the `` `index < length` `` of the quorm's multipliers.
Then in the next line the function checks if the `` `index == length` ``, which is impossible due to the above check.
When this operation is attempted to perform the exection will fail.

POC

```
    function testUpdateQuorum() public {    // POC - 0
        vm.startPrank(address(0x1337));
        IVoteWeigher.TokenMultiplier[] memory multipliers = new IVoteWeigher.TokenMultiplier[](2);
        multipliers[0] =IVoteWeigher.TokenMultiplier(address(0xdead), 100);
        multipliers[1] =IVoteWeigher.TokenMultiplier(address(0xbeef), 100);
        uint8 quorumNum = 255;

        vote.addQuorumMultiplier(quorumNum, multipliers);
        vm.expectRevert("Index out of bounds");
        //@audit Passing multipliers Length as update index.
        vote.updateQuorumMultiplier(quorumNum, multipliers.length, IVoteWeigher.TokenMultiplier(ad
dress(0x1337), 100));

    }
```

OUTPUT

```
[PASS] testUpdateQuorum() (gas: 129999)
Traces:
  [129999] VoteWeigherTest::testUpdateQuorum()
    ├─ [0] VM::startPrank(0x0000000000000000000000000000000000001337)
    │   └─ ← [Return]
    ├─ [118586] VoteWeigher::addQuorumMultiplier(255, [TokenMultiplier({ token: 0x0000000000000000
000000000000000000000dEaD, multiplier: 100 }), TokenMultiplier({ token: 0x000000000000000000000000
00000000000bEEF, multiplier: 100 })])
    │   ├─ emit QuorumAdded(quorumNumber: 255, multipliers: [TokenMultiplier({ token: 0x0000000000
000000000000000000000000dEaD, multiplier: 100 }), TokenMultiplier({ token: 0x0000000000000000000
00000000000000000bEEF, multiplier: 100 })])
    │   └─ ← [Stop]
    ├─ [0] VM::expectRevert(Index out of bounds)
    │   └─ ← [Return]
    ├─ [1215] VoteWeigher::updateQuorumMultiplier(255, 2, TokenMultiplier({ token: 0x0000000000000
000000000000000000000001337, multiplier: 100 }))
    │   └─ ← [Revert] revert: Index out of bounds
    └─ ← [Stop]

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.90ms (621.25µs CPU time)

Ran 1 test suite in 1.98s (4.90ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

biakia: In function **`updateQuorumMultiplier`**, the **`require`** statement assures the index is less than the **`quorumMultipliers[quorumNumber].length`**:

```
require(quorumMultipliers[quorumNumber].length > index, "Index out of bounds");
```

However, in the following `if` statement, it will check whether the `index` is equal to `quorumMultipliers[quorumNumber].length`:

```
if (quorumMultipliers[quorumNumber].length == index) {
            quorumMultipliers[quorumNumber].push(multiplier);
        }
```

Due to the `require` statement, the `if` statement will never be met.

Recommendation

1nc0gn170:

```
    function updateQuorumMultiplier(uint8 quorumNumber, uint256 index, TokenMultiplier memory mult
iplier) external onlyOwner {
-       require(quorumMultipliers[quorumNumber].length > index, "Index out of bounds");
+       require(quorumMultipliers[quorumNumber].length >= index, "Index out of bounds");
        if (quorumMultipliers[quorumNumber].length == index) {
            quorumMultipliers[quorumNumber].push(multiplier);
        } else {
            quorumMultipliers[quorumNumber][index] = multiplier;
        }
        emit QuorumUpdated(quorumNumber, index, multiplier);
    }
```

biakia: Consider following fix:

```
require(quorumMultipliers[quorumNumber].length >= index, "Index out of bounds");
```

Client Response

client response for 1nc0gn170: Fixed - fix by this
https://github.com/Lagrange-Labs/lagrange-contracts-audit/commit/3e68e8e4cea8f1f3e271ac6743f03b86e1a0cead
client response for biakia: Fixed - fix by this
https://github.com/Lagrange-Labs/lagrange-contracts-audit/commit/3e68e8e4cea8f1f3e271ac6743f03b86e1a0cead

LGR-5:Duplicates can increase the voteweight in `weightOfOperator`

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	1nc0gn170

Code Reference

- code/src/protocol/VoteWeigher.sol#L78-L81

```
78: for (uint256 i; i < multipliers.length; i++) {
79:             uint256 balance = stakeManager.operatorShares(operator, multipliers[i].token);
80:             totalWeight += balance * multipliers[i].multiplier;
81:         }
```

Description

1nc0gn170: The function **`weightOfOperator`** is used to calculate the voting power of an operator based on their stake amount and the multiplier in a particular quorum. However, it lacks duplicate validation, which could inflate the voting power of the operator.

POC

```
    function testDuplicateTokensWeight() public {
        vm.startPrank(address(0x1337));

        uint8 quorumNum = 255;
        // Have duplicate token
        IVoteWeigher.TokenMultiplier[] memory multipliers = new IVoteWeigher.TokenMultiplier[](2);
        multipliers[0] =IVoteWeigher.TokenMultiplier(address(token), 100);
        multipliers[1] =IVoteWeigher.TokenMultiplier(address(token), 100);


        vote.addQuorumMultiplier(quorumNum, multipliers);
        address[] memory tokens = new address[](1);
        tokens[0] = address(token);
        stake.addTokensToWhitelist(tokens);

        address user = address(0xdeadbeef);

        vm.startPrank(user);
        // Deposit 1337 ether
        token.mint(user, 1337 ether);
        token.approve(address(stake), 1337 ether);
        stake.deposit(IERC20(address(token)), 1337 ether);

        uint votingPower = vote.weightOfOperator(quorumNum, user);

        vm.assertEq(votingPower, 1337 * 2 * 100 /* Multiplier */);  // Double

    }
```

OUTPUT

```
[PASS] testDuplicateTokensWeight() (gas: 255139)
Traces:
  [255139] VoteWeigherTest::testDuplicateTokensWeight()
    ├─ [0] VM::startPrank(0x0000000000000000000000000000000000001337)
    │   └─ ← [Return]
    ├─ [118586] VoteWeigher::addQuorumMultiplier(255, [TokenMultiplier({ token: 0xF62849F9A0B5Bf29
13b396098F7c7019b51A820a, multiplier: 100 }), TokenMultiplier({ token: 0xF62849F9A0B5Bf2913b396098
F7c7019b51A820a, multiplier: 100 })])
    │   ├─ emit QuorumAdded(quorumNumber: 255, multipliers: [TokenMultiplier({ token: 0xF62849F9A0
B5Bf2913b396098F7c7019b51A820a, multiplier: 100 }), TokenMultiplier({ token: 0xF62849F9A0B5Bf2913b
396098F7c7019b51A820a, multiplier: 100 })])
    │   └─ ← [Stop]
    ├─ [25349] StakeManager::addTokensToWhitelist([0xF62849F9A0B5Bf2913b396098F7c7019b51A820a])
    │   └─ ← [Stop]
    ├─ [0] VM::startPrank(0x00000000000000000000000000000000DeaDBeef)
    │   └─ ← [Return]
```

cont.

```
    ├─ [46789] TRC20::mint(0x00000000000000000000000000000000DeaDBeef, 1337000000000000000000 [1.337e21])
    │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to: 0x00000000000000000000000000000000DeaDBeef, value: 1337000000000000000000 [1.337e21])
    │   └─ ← [Stop]
    ├─ [24739] TRC20::approve(StakeManager: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f], 1337000000000000000000 [1.337e21])
    │   ├─ emit Approval(owner: 0x00000000000000000000000000000000DeaDBeef, spender: StakeManager: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f], value: 1337000000000000000000 [1.337e21])
    │   └─ ← [Return] true
    ├─ [52395] StakeManager::deposit(TRC20: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], 1337000000000000000000 [1.337e21])
    │   ├─ [26062] TRC20::transferFrom(0x00000000000000000000000000000000DeaDBeef, StakeManager: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f], 1337000000000000000000 [1.337e21])
    │   │   ├─ emit Transfer(from: 0x00000000000000000000000000000000DeaDBeef, to: StakeManager: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f], value: 1337000000000000000000 [1.337e21])
    │   │   └─ ← [Return] true
    │   ├─ emit Deposit(operator: 0x00000000000000000000000000000000DeaDBeef, token: TRC20: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], amount: 1337000000000000000000 [1.337e21])
    │   └─ ← [Stop]
    ├─ [4742] VoteWeigher::weightOfOperator(255, 0x00000000000000000000000000000000DeaDBeef) [staticcall]
    │   ├─ [764] StakeManager::operatorShares(0x00000000000000000000000000000000DeaDBeef, TRC20: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a]) [staticcall]
    │   │   └─ ← [Return] 1337000000000000000000 [1.337e21]
    │   ├─ [764] StakeManager::operatorShares(0x00000000000000000000000000000000DeaDBeef, TRC20: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a]) [staticcall]
    │   │   └─ ← [Return] 1337000000000000000000 [1.337e21]
    │   └─ ← [Return] 267400 [2.674e5]
    ├─ [0] VM::assertEq(267400 [2.674e5], 267400 [2.674e5]) [staticcall]
    │   └─ ← [Return]
    └─ ← [Stop]
```

Recommendation

1nc0gn170: Either don't allow duplicate tokens in `TokenMultiplier` or do filtering while calculating voting power.

```
    function weightOfOperator(uint8 quorumNumber, address operator)
        external
        view
        returns (uint96)
    {
        uint256 totalWeight = 0;
-       TokenMultiplier[] memory multipliers = quorumMultipliers[quorumNumber];
+       TokenMultiplier[] memory multipliers = /* Get Unique Token Multipliers */
        for (uint256 i; i < multipliers.length; i++) {
            uint256 balance = stakeManager.operatorShares(operator, multipliers[i].token);
            totalWeight += balance * multipliers[i].multiplier;
        }
        return uint96(totalWeight / WEIGHTING_DIVISOR);
    }
```

Client Response

client response for 1nc0gn170: Fixed - . changed severity to Low .Since it is only limited to owner's mistakes, furthermore it is possible to recover by removeQuorumMultiplier / updateQuorumMultiplier.
fix by this
https://github.com/Lagrange-Labs/lagrange-contracts-audit/commit/3e68e8e4cea8f1f3e271ac6743f03b86e1a0cead
Secure3: changed severity to Low .Since it is only limited to owner’s mistakes, furthermore it is possible to recover by removeQuorumMultiplier / updateQuorumMultiplier.

LGR-6:missing indexed keyword in event log

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	newway55

Code Reference

- code/src/protocol/VoteWeigher.sol#L24C4-L26C88

```
NaN: event QuorumAdded(uint8 quorumNumber, TokenMultiplier[] multipliers);
NaN:     event QuorumRemoved(uint8 quorumNumber);
NaN:     event QuorumUpdated(uint8 quorumNumber, uint256 index, TokenMultiplier multiplier);
```

Description

newway55: #### Description
The smart contract's design omits indexed parameters in key events (**`QuorumAdded`**, **`QuorumRemoved`**, and **`QuorumUpdated`**), leading to a less efficient mechanism for querying event logs. Indexed parameters are crucial for facilitating effective and fast searches within Ethereum's log data structure, improving the ability of off-chain applications to identify and react to specific contract events. The absence of indexed parameters necessitates a full scan of all emitted events to locate relevant entries, impacting the efficiency of data retrieval processes.

POC

Consider a scenario where an off-chain application monitors **`QuorumAdded`** events for a specific **`quorumNumber`**. Without **`quorumNumber`** being indexed, the application must retrieve every **`QuorumAdded`** event and **iterate** through them to find events of interest. This process is significantly less efficient than directly querying for events with a specific **`quorumNumber`**, which is only feasible if **`quorumNumber`** is **indexed**.

Recommendation

newway55: It is recommended to modify the event definitions to include indexed parameters for key identifiers as following :

```
event QuorumAdded(uint8 indexed quorumNumber, TokenMultiplier[] multipliers);
event QuorumRemoved(uint8 indexed quorumNumber);
event QuorumUpdated(uint8 indexed quorumNumber, uint256 indexed index, TokenMultiplier multiplie
r);
```

By indexing **`quorumNumber`** (and index in the **`QuorumUpdated`** event), off-chain applications can efficiently filter the logs for events to specific quorums improve the efficiency of apps.

Client Response

client response for newway55: Fixed - fix by this
https://github.com/Lagrange-Labs/lagrange-contracts-audit/commit/3e68e8e4cea8f1f3e271ac6743f03b86e1a0cead

LGR-7:Using `calldata` instead of `memory` for read-only arguments in external functions saves gas.

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	n16h7m4r3

Code Reference

- code/src/protocol/LagrangeCommittee.sol#L61
- code/src/protocol/LagrangeCommittee.sol#L67

```
61: function addOperator(address operator, address signAddress, uint256[2][] memory blsPubKeys) publ
ic onlyService {
```

```
67: function addBlsPubKeys(address operator, uint256[2][] memory additionalBlsPubKeys) public onlySe
rvice {
```

- code/src/protocol/LagrangeService.sol#L66
- code/src/protocol/LagrangeService.sol#L78
- code/src/protocol/LagrangeService.sol#L114

```
66: uint256[2][] memory blsPubKeys,
```

```
78: function addBlsPubKeys(uint256[2][] memory additionalBlsPubKeys) external onlyWhitelisted {
```

```
114: function updateAVSMetadataURI(string memory _metadataURI) public virtual onlyOwner {
```

- code/src/protocol/VoteWeigher.sol#L38
- code/src/protocol/VoteWeigher.sol#L89

```
38: function addQuorumMultiplier(uint8 quorumNumber, TokenMultiplier[] memory multipliers) external
onlyOwner {
```

```
89: function getTokenListForQuorumNumbers(uint8[] memory quorumNumbers_) external view returns (addr
ess[] memory) {
```

Description

n16h7m4r3: When a function with a **`memory`** array is called externally, the **`abi.decode()`** step has to use a for-loop to copy each index of the **`calldata`** to the **`memory`** index. Each iteration of this for-loop costs at least 60 gas (i.e. **`60 * <mem_array>.length`**). Using **`calldata`** directly, obliviates the need for such a loop in the contract code and runtime execution.
If the array is passed to an **`internal`** function which passes the array to another **`internal`** function where the array is modified and therefore **`memory`** is used in the external call, it's still more gas-efficient to use calldata when the external function uses modifiers, since the modifiers may prevent the **`internal`** functions from being called. Structs have the same overhead as an array of length one.

Recommendation

n16h7m4r3: Consider using `` `calldata` `` instead of `` `memory` ``.

Client Response

client response for n16h7m4r3: Fixed - fix by this
https://github.com/Lagrange-Labs/lagrange-contracts-audit/commit/3e68e8e4cea8f1f3e271ac6743f03b86e1a0cead

Disclaimer