

Started	Sun Jan 22 2023 14:53:30 GMT+0000 (Coordinated Universal Time)
Finished	Sun Jan 22 2023 15:03:34 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Remythx
Main Source File	Entry_flat.sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	5

ISSUES

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
12 | address(this),
13 | id,
14 | blockhash(block.number - 1),
15 | block.difficulty,
16 | block.timestamp,
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
21 |
22 | function _gasPrice() internal view returns (uint256) {
23 |     uint256 maxFee = block.basefee + (block.basefee / 4);
24 |     uint256 gasPrice = tx.gasprice < maxFee ? tx.gasprice : maxFee;
25 |     return gasPrice;
```

UNKNOWN Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
21 |
22 | function _gasPrice() internal view returns (uint256) {
23 |     uint256 maxFee = block.basefee + (block.basefee / 4);
24 |     uint256 gasPrice = tx.gasprice < maxFee ? tx.gasprice : maxFee;
25 |     return gasPrice;
```

UNKNOWN Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
70 | uint256 t;
71 | while (_x != 0) {
72 |     t = r / _x;
73 |     (q, newT) = (newT, addmod(q, (PP - mulmod(t, newT, PP)), PP));
74 |     (r, _x) = (_x, r - t * _x);
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
71 | while (_x != 0) {
72 |     t = r / _x;
73 |     (q, newT) = (newT, addmod(q, (PP - mulmod(t, newT, PP)), PP));
74 |     (r, _x) = (_x, r - t * _x);
75 | }
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
72 | t = r / _x;  
73 | (q, newT) = (newT, addmod(q, (PP - mulmod(t, newT, PP)), PP));  
74 | (r, _x) = (_x, r - t*_x);  
75 | }  
76 |
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
72 | t = r / _x;  
73 | (q, newT) = (newT, addmod(q, (PP - mulmod(t, newT, PP)), PP));  
74 | (r, _x) = (_x, r - t*_x);  
75 | }  
76 |
```

UNKNOWN Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
135 | // x^3 + ax + b  
136 | uint256 y2 = addmod(mulmod(_x, mulmod(_x, _x, PP), PP), addmod(mulmod(_x, AA, PP), BB, PP), PP);  
137 | y2 = expMod(y2, [PP+1]/4);  
138 | // uint256 cmp = yBit ^ y_ & 1;  
139 | uint256 y = (y2 + _prefix) % 2 == 0 ? y2 : PP - y2;
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
135 | // x^3 + ax + b
136 | uint256 y2 = addmod(mulmod(_x, mulmod(_x, _x, PP), PP), addmod(mulmod(_x, AA, PP), BB, PP), PP);
137 | y2 = expMod(y2, (PP + 1) / 4);
138 | // uint256 cmp = yBit ^ y_ & 1;
139 | uint256 y = (y2 + _prefix) % 2 == 0 ? y2 : PP - y2;
```

UNKNOWN Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
137 | y2 = expMod(y2, (PP + 1) / 4);
138 | // uint256 cmp = yBit ^ y_ & 1;
139 | uint256 y = (y2 + _prefix) % 2 == 0 ? y2 : PP - y2;
140 |
141 | return y;
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
137 | y2 = expMod(y2, (PP + 1) / 4);
138 | // uint256 cmp = yBit ^ y_ & 1;
139 | uint256 y = (y2 + _prefix) % 2 == 0 ? y2 : PP - y2;
140 |
141 | return y;
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
137 | y2 = expMod(y2, (PP + 1) / 4);
138 | // uint256 cmp = y8bit ^ y_ & 1;
139 | uint256 y = (y2 + _prefix) % 2 == 0 ? y2 : PP - y2;
140 |
141 | return y;
```

UNKNOWN Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
171 | /// @return (x, -y)
172 | function ecInv(uint256 _x, uint256 _y) internal pure returns (uint256, uint256) {
173 |     return (_x, PP - _y % PP);
174 | }
175 |
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
171 | /// @return (x, -y)
172 | function ecInv(uint256 _x, uint256 _y) internal pure returns (uint256, uint256) {
173 |     return (_x, (PP - _y) % PP);
174 | }
175 |
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
283 | uint256[4] memory hr;  
284 |  
285 | hr[0] = addmod(zs[2], PP - zs[0], PP);  
286 |  
287 | hr[1] = addmod(zs[3], PP - zs[1], PP);
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
285 | hr[0] = addmod(zs[2], PP - zs[0], PP);  
286 |  
287 | hr[1] = addmod(zs[3], PP - zs[1], PP);  
288 |  
289 | hr[2] = mulmod(hr[0], hr[0], PP);
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
291 | hr[3] = mulmod(hr[2], hr[0], PP);  
292 | // qx = -h^3 -2u1h^2+r^2  
293 | uint256 qx = addmod(mulmod(hr[1], hr[1], PP), PP - hr[3], PP);  
294 | qx = addmod(qx, PP - mulmod(2, mulmod(zs[0], hr[2], PP), PP), PP);  
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
292 | // qx = -h^3 -2u1h^2+r^2
293 | uint256 qx = addmod(mulmod(hr[1], hr[1], PP), PP - hr[3], PP);
294 | qx = addmod(qx, PP - mulmod(2, mulmod(zs[0], hr[2], PP), PP), PP);
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
296 | uint256 qy = mulmod(hr[1], addmod(mulmod(zs[0], hr[2], PP), PP - qx, PP), PP);
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
294 | qx = addmod(qx, PP - mulmod(2, mulmod(zs[0], hr[2], PP), PP), PP);
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
296 | uint256 qy = mulmod(hr[1], addmod(mulmod(zs[0], hr[2], PP), PP - qx, PP), PP);
297 | qy = addmod(qy, PP - mulmod(zs[1], hr[3], PP), PP);
298 | // qz = h*z1*z2
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
296 | uint256 qy = mulmod(hr[1], addmod(mulmod(zs[0], hr[2], PP), PP - qx, PP), PP);
297 | qy = addmod(qy, PP - mulmod(zs[1], hr[3], PP), PP);
298 | // qz = h*z1*z2
299 | uint256 qz = mulmod(hr[0], mulmod(_z1, _z2, PP), PP);
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
339 | // This allows to reduce the gas cost and stack footprint of the algorithm
340 | // qx
341 | x = addmod(mulmod(m, m, PP), PP - addmod(s, s, PP), PP);
342 | // qy = -8*y1^4 + M(S-T)
343 | y = addmod(mulmod(m, addmod(s, PP - x, PP), PP), PP - mulmod(8, mulmod(y, y, PP), PP), PP);
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
341 | x = addmod(mulmod(m, m, PP), PP - addmod(s, s, PP), PP);
342 | // qy = -8*y1^4 + M(S-T)
343 | y = addmod(mulmod(m, addmod(s, PP - x, PP), PP), PP - mulmod(8, mulmod(y, y, PP), PP), PP);
344 | // qz = 2*y1*z1
345 | z = mulmod(2, mulmod(_y, _z, PP), PP);
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
341 | x = addmod(mulmod(m, m, PP), PP - addmod(s, s, PP), PP);
342 | // qy = -8*y1^4 + M(S-T)
343 | y = addmod(mulmod(m, addmod(s, PP - x, PP), PP), PP - mulmod(8, mulmod(y, y, PP), PP), PP);
344 | // qz = 2*y1*z1
345 | z = mulmod(2, mulmod(_y, _z, PP), PP);
```


UNKNOWN Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
383 | (qx, qy, qz) = jacAdd(qx, qy, qz, _x, _y, _z);
384 | }
385 | remaining = remaining / 2;
386 | (_x, _y, _z) = jacDouble(_x, _y, _z);
387 | }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
680 | }
681 | unchecked {
682 |     ++ctr;
683 | }
684 | while (ctr < 256);
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
735 | /// @return The point coordinates as bytes
736 | function encodePoint(uint256 _x, uint256 _y) internal pure returns (bytes memory) {
737 |     uint8 prefix = uint8(2 + (_y % 2));
738 |
739 |     return abi.encodePacked(prefix, _x);
```

UNKNOWN Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
735 | /// @return The point coordinates as bytes
736 | function encodePoint(uint256 _x, uint256 _y) internal pure returns (bytes memory) {
737 |     uint8 prefix = uint8(2 + (_y % 2));
738 |
739 |     return abi.encodePacked(prefix, _x);
```

UNKNOWN Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
779 | uint256 _qy
780 | ) internal pure returns (bool) {
781 |     address result = ecrecover(0, _y % 2 != 0 ? 28 : 27, bytes32(_x), bytes32(mulmod(_scalar, _x, NN)));
782 |
783 |     return pointToAddress(_qx, _qy) == result;
```

UNKNOWN Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
882 | uint256 _qy
883 | ) internal pure returns (bool) {
884 |     uint256 scalar1 = (NN - _scalar1) % NN;
885 |     scalar1 = mulmod(scalar1, _x, NN);
886 |     uint256 scalar2 = (NN - _scalar2) % NN;
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
802 | uint256 _qy
803 | ) internal pure returns (bool) {
804 | uint256 scalar1 = (NN - _scalar1) % NN;
805 | scalar1 = mulmod(scalar1, _x, NN);
806 | uint256 scalar2 = (NN - _scalar2) % NN;
```

UNKNOWN Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
804 | uint256 scalar1 = (NN - _scalar1) % NN;
805 | scalar1 = mulmod(scalar1, _x, NN);
806 | uint256 scalar2 = (NN - _scalar2) % NN;
807 |
808 | address result = ecrecover(
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
804 | uint256 scalar1 = (NN - _scalar1) % NN;
805 | scalar1 = mulmod(scalar1, _x, NN);
806 | uint256 scalar2 = (NN - _scalar2) % NN;
807 |
808 | address result = ecrecover(
```

UNKNOWN Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
808 | address result = ecrecover(  
809 | bytes32(scalar1),  
810 | _y%2 != 0 ? 28 : 27,  
811 | bytes32(_x),  
812 | bytes32(mod(scalar2, _x, NN))
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1027 | uint256 offset  
1028 | ) internal view returns (uint256) {  
1029 | return _beaconFee + (LibNetwork._gasPrice() * gasAtStart + offset - gasleft());  
1030 | }  
1031 |
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1027 | uint256 offset  
1028 | ) internal view returns (uint256) {  
1029 | return _beaconFee + (LibNetwork._gasPrice() * gasAtStart + offset - gasleft());  
1030 | }  
1031 |
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1027 | uint256 offset
1028 | ) internal view returns (uint256) {
1029 | return _beaconFee + (LibNetwork._gasPrice() * (gasAtStart + offset - gasleft()));
1030 | }
1031 |
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1027 | uint256 offset
1028 | ) internal view returns (uint256) {
1029 | return _beaconFee + (LibNetwork._gasPrice() * (gasAtStart + offset - gasleft()));
1030 | }
1031 |
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1326 | bytes memory _calldata
1327 | ) internal {
1328 | for (uint256 facetIndex; facetIndex < _diamondCut.length; facetIndex++) {
1329 | IDiamondCut.FacetCutAction action = _diamondCut[facetIndex].action;
1330 | if (action == IDiamondCut.FacetCutAction.Add) {
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1357 | addFacet(ds, _facetAddress);
1358 | }
1359 | for (uint256 selectorIndex; selectorIndex < _functionSelectors.length; selectorIndex++) {
1360 |     bytes4 selector = _functionSelectors[selectorIndex];
1361 |     address oldFacetAddress = ds.selectorToFacetAndPosition[selector].facetAddress;
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1362 | require(oldFacetAddress == address(0), "LibDiamondCut: Can't add function that already exists");
1363 | addFunction(ds, selector, selectorPosition, _facetAddress);
1364 | selectorPosition++;
1365 | }
1366 | }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1375 | addFacet(ds, _facetAddress);
1376 | }
1377 | for (uint256 selectorIndex; selectorIndex < _functionSelectors.length; selectorIndex++) {
1378 |     bytes4 selector = _functionSelectors[selectorIndex];
1379 |     address oldFacetAddress = ds.selectorToFacetAndPosition[selector].facetAddress;
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1384 | removeFunction(ds, oldFacetAddress, selector);
1385 | addFunction(ds, selector, selectorPosition, _facetAddress);
1386 | selectorPosition++;
1387 | }
1388 | }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1393 | // if function does not exist then do nothing and return
1394 | require(_facetAddress == address(0), "LibDiamondCut: Remove facet address must be address(0)");
1395 | for (uint256 selectorIndex; selectorIndex < _functionSelectors.length; selectorIndex++) {
1396 |     bytes4 selector = _functionSelectors[selectorIndex];
1397 |     address oldFacetAddress = ds.selectorToFacetAndPosition[selector].facetAddress;
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1427 | // replace selector with last selector, then delete last selector
1428 | uint256 selectorPosition = ds.selectorToFacetAndPosition[_selector].functionSelectorPosition;
1429 | uint256 lastSelectorPosition = ds.facetFunctionSelectors[_facetAddress].functionSelectors.length - 1;
1430 | // if not the same then replace _selector with lastSelector
1431 | if (selectorPosition != lastSelectorPosition) {
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1443 | if (lastSelectorPosition == 0) {  
1444 | // replace facet address with last facet address and delete last facet address  
1445 | uint256 lastFacetAddressPosition = ds.facetAddresses.length - 1;  
1446 | uint256 facetAddressPosition = ds.facetFunctionSelectors[_facetAddress].facetAddressPosition;  
1447 | if (facetAddressPosition != lastFacetAddressPosition) {
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1590 | uint256 index = s.beaconIndex[_beacon];  
1591 | if (index == 0) revert BeaconNotFound();  
1592 | uint256 lastBeaconIndex = s.beacons.length - 1;  
1593 | s.beacon[_beacon].registered = false;  
1594 | if (index == lastBeaconIndex) {
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1616 | _data.timestamp = block.timestamp;  
1617 | address randomBeacon = _selectOneBeacon(_seed, [_accounts.beacons[0], _accounts.beacons[1]]);  
1618 | s.beacon[randomBeacon].pending++;  
1619 | _accounts.beacons[_beaconPos] = randomBeacon;  
1620 | s.requestToHash[_id] = LibBeacon._generateRequestHash(_id, _accounts, _data, _seed);
```


UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1630 | do {
1631 | // Generate a random index j such that i <= j <= selectedItems.length - 1
1632 | uint256 j = (uint256(keccak256(abi.encodePacked(_random, i))) % selectedItems.length - i) + i;
1633 | // Swap the items at indices i and j
1634 | address temp = selectedItems[i];
```

UNKNOWN Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1630 | do {
1631 | // Generate a random index j such that i <= j <= selectedItems.length - 1
1632 | uint256 j = ((uint256(keccak256(abi.encodePacked(_random, i))) % selectedItems.length - i) + i;
1633 | // Swap the items at indices i and j
1634 | address temp = selectedItems[i];
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1630 | do {
1631 | // Generate a random index j such that i <= j <= selectedItems.length - 1
1632 | uint256 j = (uint256(keccak256(abi.encodePacked(_random, i))) % (selectedItems.length - i)) + i;
1633 | // Swap the items at indices i and j
1634 | address temp = selectedItems[i];
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1635 | selectedItems[i] = selectedItems[j];
1636 | selectedItems[j] = temp;
1637 | s.beacon[selectedItems[i]].pending++;
1638 | unchecked {
1639 | ++i;
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1637 | s.beacon[selectedItems[i]].pending++;
1638 | unchecked {
1639 | ++i;
1640 | }
1641 | while (i < 3);
```

UNKNOWN Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1649 |
1650 | // Generate a random index j such that j <= count
1651 | uint256 j = uint256(_random) % count;
1652 |
1653 | return selectedItems[j];
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1665 | {
1666 |     uint256 beaconsLen = s.beacons.length;
1667 |     address[] memory selectedItems = new address[](beaconsLen - 2);
1668 |
1669 |     uint256 i = 1;
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1677 | }
1678 | unchecked {
1679 |     ++j;
1680 | }
1681 | }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1683 | selectedItems[count] = s.beacons[i];
1684 | unchecked {
1685 |     ++count;
1686 | }
1687 | }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1687 | }
1688 | unchecked {
1689 |   ++i;
1690 | }
1691 | while (i < beaconsLen);
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1700 | {
1701 |   uint256 beaconsLen = s.beacons.length;
1702 |   address[] memory selectedItems = new address[](beaconsLen - 3);
1703 |
1704 |   uint256 i = 1;
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1712 | }
1713 | unchecked {
1714 |   ++j;
1715 | }
1716 | }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1718 | selectedItems[count] = s.beacons[i];
1719 | unchecked {
1720 |   ++count;
1721 | }
1722 | }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1722 | }
1723 | unchecked {
1724 |   i++;
1725 | }
1726 | while (i < beaconsLen);
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1735 | {
1736 |   uint256 beaconsLen = s.beacons.length;
1737 |   address[] memory selectedItems = new address[](beaconsLen - excludeLen);
1738 |
1739 |   uint256 i = 1;
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
1747 | }  
1748 | unchecked {  
1749 | ++j;  
1750 | }  
1751 | }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
1753 | selectedItems[count] = s.beacons[i];  
1754 | unchecked {  
1755 | ++count;  
1756 | }  
1757 | }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
1757 | }  
1758 | unchecked {  
1759 | ++i;  
1760 | }  
1761 | while (i < beaconsLen);
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1775 | // Callback to requesting contract
1776 | LibBeacon._callback(client, callbackGasLimit, id, result);
1777 | s.ethReserved(client) -= _ethReserved;
1778 |
1779 | s.results[id] = result;
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1795 | // If this is the final charge for the request,
1796 | // add fee for configured treasury and sequencer
1797 | daoFee = deposit >= fee + beaconFee ? beaconFee : deposit - fee;
1798 | _chargeClient(client, s.treasury, daoFee);
1799 | // Only add sequencer fee if the deposit has enough subtracting sender and treasury fee
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1795 | // If this is the final charge for the request,
1796 | // add fee for configured treasury and sequencer
1797 | daoFee = deposit >= fee + beaconFee ? beaconFee : deposit - fee;
1798 | _chargeClient(client, s.treasury, daoFee);
1799 | // Only add sequencer fee if the deposit has enough subtracting sender and treasury fee
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1798 | _chargeClient(client, s.treasury, daoFee);
1799 | // Only add sequencer fee if the deposit has enough subtracting sender and treasury fee
1800 | if (deposit > fee + daoFee) {
1801 |     seqFee = deposit >= fee + daoFee + beaconFee ? beaconFee : deposit - daoFee - fee;
1802 |     _chargeClient(client, s.sequencer, seqFee);
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1799 | // Only add sequencer fee if the deposit has enough subtracting sender and treasury fee
1800 | if (deposit > fee + daoFee) {
1801 |     seqFee = deposit >= fee + daoFee + beaconFee ? beaconFee : deposit - daoFee - fee;
1802 |     _chargeClient(client, s.sequencer, seqFee);
1803 | }
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1799 | // Only add sequencer fee if the deposit has enough subtracting sender and treasury fee
1800 | if (deposit > fee + daoFee) {
1801 |     seqFee = deposit >= fee + daoFee + beaconFee ? beaconFee : deposit - daoFee - fee;
1802 |     _chargeClient(client, s.sequencer, seqFee);
1803 | }
```


UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1799 | // Only add sequencer fee if the deposit has enough subtracting sender and treasury fee
1800 | if (deposit > fee + daoFee) {
1801 |     seqFee = deposit >= fee + daoFee + beaconFee ? beaconFee : deposit - daoFee - fee;
1802 |     _chargeClient(client, s.sequencer, seqFee);
1803 | }
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1799 | // Only add sequencer fee if the deposit has enough subtracting sender and treasury fee
1800 | if (deposit > fee + daoFee) {
1801 |     seqFee = deposit >= fee + daoFee + beaconFee ? beaconFee : deposit - daoFee - fee;
1802 |     _chargeClient(client, s.sequencer, seqFee);
1803 | }
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1805 | fee = deposit;
1806 | }
1807 | s.requestToFeePaid(id) += fee + seqFee + daoFee;
1808 | _chargeClient(client, msg.sender, fee);
1809 | }
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1895 | fee = deposit;
1896 | }
1897 | s.requestToFeePaid[id] += fee + seqFee + daoFee;
1898 | _chargeClient(client, msg.sender, fee);
1899 | }
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1895 | fee = deposit;
1896 | }
1897 | s.requestToFeePaid[id] += fee + seqFee + daoFee;
1898 | _chargeClient(client, msg.sender, fee);
1899 | }
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1821 | fee = deposit;
1822 | }
1823 | s.requestToFeePaid[id] += fee;
1824 | _chargeClient(client, msg.sender, fee);
1825 | }
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1840 | uint256 _value
1841 | ) private {
1842 | s.ethDeposit[_from] -= _value;
1843 | s.ethCollateral[_to] += _value;
1844 | emit Events.ChargeEth(_from, _to, _value, Constants.CHARGE_TYPE_CLIENT_TO_BEACON);
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1841 | ) private {
1842 | s.ethDeposit[_from] -= _value;
1843 | s.ethCollateral[_to] += _value;
1844 | emit Events.ChargeEth(_from, _to, _value, Constants.CHARGE_TYPE_CLIENT_TO_BEACON);
1845 | }
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1942 | ) external {
1943 | // 20k gas offset for balance updates after fee calculation
1944 | uint256 gasAtStart = gasleft() + s.gasEstimates[Constants.GKEY_OFFSET_RENEW];
1945 |
1946 | SAccounts memory accounts = LibBeacon._resolveAddressCalldata(_addressData);
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1961 |  
1962 | {  
1963 | uint256 _expirationHeight = packed.data.height + packed.data.expirationBlocks;  
1964 | uint256 _expirationTime = packed.data.timestamp + packed.data.expirationSeconds;  
1965 | if (msg.sender == s.sequencer) {
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1962 | {  
1963 | uint256 _expirationHeight = packed.data.height + packed.data.expirationBlocks;  
1964 | uint256 _expirationTime = packed.data.timestamp + packed.data.expirationSeconds;  
1965 | if (msg.sender == s.sequencer) {  
1966 | _expirationHeight += packed.data.expirationBlocks / 2;
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1964 | uint256 _expirationTime = packed.data.timestamp + packed.data.expirationSeconds;  
1965 | if (msg.sender == s.sequencer) {  
1966 | _expirationHeight += packed.data.expirationBlocks / 2;  
1967 | _expirationTime += packed.data.expirationSeconds / 2;  
1968 | } else if (
```

UNKNOWN Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1964 | uint256 _expirationTime = packed.data.timestamp + packed.data.expirationSeconds;  
1965 | if (msg.sender == s.sequencer) {  
1966 |     _expirationHeight += packed.data.expirationBlocks / 2;  
1967 |     _expirationTime += packed.data.expirationSeconds / 2;  
1968 | } else if (
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1965 | if (msg.sender == s.sequencer) {  
1966 |     _expirationHeight += packed.data.expirationBlocks / 2;  
1967 |     _expirationTime += packed.data.expirationSeconds / 2;  
1968 | } else if (  
1969 |     // First beacon can renew first if they submitted
```

UNKNOWN Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1965 | if (msg.sender == s.sequencer) {  
1966 |     _expirationHeight += packed.data.expirationBlocks / 2;  
1967 |     _expirationTime += packed.data.expirationSeconds / 2;  
1968 | } else if (  
1969 |     // First beacon can renew first if they submitted
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1973 | (msg.sender == accounts.beacons[1] && hashes[1] != bytes10(0) && hashes[0] == bytes10(0))
1974 | ) {
1975 |     _expirationHeight += packed.data.expirationBlocks;
1976 |     _expirationTime += packed.data.expirationSeconds;
1977 | }
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1974 | ) {
1975 |     _expirationHeight += packed.data.expirationBlocks;
1976 |     _expirationTime += packed.data.expirationSeconds;
1977 | }
1978 |
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1990 | uint8 beaconsToStrikeLen = 0;
1991 | address[3] memory reqBeacons = accounts.beacons;
1992 | for (uint256 i; i < 2; i++) {
1993 |     if (hashes[i] == bytes10(0) && reqBeacons[i] != address(0)) {
1994 |         address beaconAddress = reqBeacons[i];
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1995 | _strikeBeacon(beaconAddress);
1996 | beaconsToStrike[i] = beaconAddress;
1997 | beaconsToStrikeLen++;
1998 | }
1999 | }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2006 | _strikeBeacon(beaconAddress);
2007 | beaconsToStrike[2] = beaconAddress;
2008 | beaconsToStrikeLen++;
2009 | }
2010 | }
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2010 |
2011 | // Checks if enough beacons are available to replace with
2012 | if (s.beacons.length < 5 || beaconsToStrikeLen * 2 > s.beacons.length - 1)
2013 | revert NotEnoughBeaconsAvailable(
2014 | s.beacons.length,
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2010 |  
2011 | // Checks if enough beacons are available to replace with  
2012 | if (s.beacons.length < 5 || beaconsToStrikeLen * 2 > s.beacons.length - 1)  
2013 | revert NotEnoughBeaconsAvailable(  
2014 | s.beacons.length,
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2013 | revert NotEnoughBeaconsAvailable(  
2014 | s.beacons.length,  
2015 | s.beacons.length < 5 ? 5 : beaconsToStrikeLen * 2  
2016 | );  
2017 |
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2021 | // Add gas fee for refund function  
2022 | address firstStrikeBeacon;  
2023 | for (uint256 i; i < beaconsToStrike.length; i++) {  
2024 | if (beaconsToStrike[i] == address(0)) continue;  
2025 |
```


UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2062 |
2063 | // The paying non-submitter might fall below collateral here. It will be removed on next strike if it doesn't add collateral.
2064 | uint256 renewFee = packed.data.beaconFee + (LibNetwork.gasPrice() * (gasAtStart - gasleft()));
2065 |
2066 | uint256 refundToClient = s.requestToFeePaid[packed.id];
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2062 |
2063 | // The paying non-submitter might fall below collateral here. It will be removed on next strike if it doesn't add collateral.
2064 | uint256 renewFee = packed.data.beaconFee + (LibNetwork.gasPrice() * (gasAtStart - gasleft()));
2065 |
2066 | uint256 refundToClient = s.requestToFeePaid[packed.id];
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2062 |
2063 | // The paying non-submitter might fall below collateral here. It will be removed on next strike if it doesn't add collateral.
2064 | uint256 renewFee = packed.data.beaconFee + (LibNetwork.gasPrice() * (gasAtStart - gasleft()));
2065 |
2066 | uint256 refundToClient = s.requestToFeePaid[packed.id];
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2065 |
2066 | uint256 refundToClient = s.requestToFeePaid[packed.id];
2067 | uint256 totalCharge = renewFee + refundToClient;
2068 |
2069 | // If charging more than the striked beacon has staked, refund the remaining stake to the client
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2073 | totalCharge = firstCollateral;
2074 | renewFee = renewFee > totalCharge ? totalCharge : renewFee;
2075 | s.ethCollateral(msg.sender) += renewFee;
2076 | emit Events.ChargeEth(
2077 | firstStrikeBeacon,
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2082 | // totalCharge - renewFee is now 0 at its lowest
2083 | // If collateral is remaining after renewFee, it will be refunded to the client
2084 | refundToClient = totalCharge - renewFee;
2085 | if (refundToClient > 0) {
2086 | s.ethDeposit[accounts.client] += refundToClient;
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
2084 | refundToClient = totalCharge - renewFee;
2085 | if (refundToClient > 0) {
2086 |   s.ethDeposit(accounts.client) += refundToClient;
2087 |   emit Events.ChargeEth(
2088 |     firstStrikeBeacon,
```

UNKNOWN Arithmetic operation "-=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
2094 | s.ethCollateral[firstStrikeBeacon] = 0;
2095 | } else {
2096 |   s.ethCollateral[firstStrikeBeacon] -= totalCharge;
2097 |   // Refund this function's gas to the caller
2098 |   s.ethCollateral[msg.sender] += renewFee;
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
2096 | s.ethCollateral[firstStrikeBeacon] -= totalCharge;
2097 | // Refund this function's gas to the caller
2098 | s.ethCollateral[msg.sender] += renewFee;
2099 | s.ethDeposit(accounts.client) += refundToClient;
2100 | // Add to fees refunded
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2097 | // Refund this function's gas to the caller
2098 | s.ethCollateral[msg.sender] += renewFee;
2099 | s.ethDeposit[accounts.client] += refundToClient;
2100 | // Add to fees refunded
2101 | s.requestToFeeRefunded[packed.id] += refundToClient;
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2099 | s.ethDeposit[accounts.client] += refundToClient;
2100 | // Add to fees refunded
2101 | s.requestToFeeRefunded[packed.id] += refundToClient;
2102 | // Client receives refund to ensure they have enough to pay for the next request
2103 | // Also since the request is taking slower than expected due to a non-submitting beacon,
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2124 | function _strikeBeacon(address _beacon) internal {
2125 |     Beacon memory tempBeacon = s.beacon[_beacon];
2126 |     if (tempBeacon.registered) tempBeacon.strikes++;
2127 |     tempBeacon.consecutiveSubmissions = 0;
2128 |     if (tempBeacon.pending > 0) tempBeacon.pending--;
```

UNKNOWN Arithmetic operation "--" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2126 | if (tempBeacon.registered) tempBeacon.strikes++;
2127 | tempBeacon.consecutiveSubmissions = 0;
2128 | if (tempBeacon.pending > 0) tempBeacon.pending--;
2129 | s.beacon[_beacon] = tempBeacon;
2130 | }
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2142 | address(this),
2143 | _request,
2144 | LibNetwork._blockHash(LibNetwork.blockNumber()-1),
2145 | block.chainid
2146 | )
```

UNKNOWN Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2161 | ) {
2162 | // Generate new beacon beacon index
2163 | uint256 randomBeaconIndex = uint256(random)%count;
2164 | // Get a random beacon from the available beacons
2165 | address randomBeacon = availableBeacons[randomBeaconIndex];
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2166 | // Assign the random beacon to newSelectedBeacons
2167 | newSelectedBeacons[i] = randomBeacon;
2168 | s.beacon[randomBeacon].pending++;
2169 | // Add the beacon to the excluded beacons
2170 | excludedBeacons[excludedBeaconCount] = randomBeacon;
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2169 | // Add the beacon to the excluded beacons
2170 | excludedBeacons[excludedBeaconCount] = randomBeacon;
2171 | excludedBeaconCount++;
2172 | // Update the available beacons
2173 | (availableBeacons, count) = _beaconsWithoutExcluded(excludedBeacons, excludedBeaconCount);
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2177 | }
2178 | unchecked {
2179 |     i++;
2180 | }
2181 | }
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2243  /// @param _client The address of the client contract to deposit ETH to
2244  function clientDeposit(address _client) external payable {
2245      s.ethDeposit[_client] += msg.value;
2246      emit Events.ClientDepositEth(_client, msg.value);
2247  }
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2254  function clientWithdrawTo(address _to, uint256 _amount) external {
2255      // Check if the client is trying to withdraw more than they have deposited
2256      if (_amount > s.ethDeposit[msg.sender] - s.ethReserved[msg.sender])
2257          revert WithdrawingTooMuch(_amount, s.ethDeposit[msg.sender] - s.ethReserved[msg.sender]);
2258  }
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2255  // Check if the client is trying to withdraw more than they have deposited
2256  if (_amount > s.ethDeposit[msg.sender] - s.ethReserved[msg.sender])
2257      revert WithdrawingTooMuch(_amount, s.ethDeposit[msg.sender] - s.ethReserved[msg.sender]);
2258
2259  // Decrease the client's deposit by the amount they are withdrawing
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2258 |
2259 | // Decrease the client's deposit by the amount they are withdrawing
2260 | s.ethDeposit(msg.sender) -= _amount;
2261 |
2262 | // Emit an event to log the withdrawal
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2273 | function estimateFee(uint256 _callbackGasLimit) public view returns (uint256 estimateFee) {
2274 |     return
2275 |         (s.gasEstimates.Constants.GKEY_TOTAL_SUBMIT) +
2276 |         _callbackGasLimit +
2277 |         (s.gasEstimates.Constants.GKEY_GAS_PER_BEACON_SELECT * (s.beacons.length - 1) * 3) *
2278 |         LibNetwork.gasPrice() + (s.configUints.Constants.CKEY_BEACON_FEE * 5);
2279 | }
2280 |
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2273 | function estimateFee(uint256 _callbackGasLimit) public view returns (uint256 estimateFee) {
2274 |     return
2275 |         (s.gasEstimates.Constants.GKEY_TOTAL_SUBMIT) +
2276 |         _callbackGasLimit +
2277 |         (s.gasEstimates.Constants.GKEY_GAS_PER_BEACON_SELECT * (s.beacons.length - 1) * 3) *
2278 |         LibNetwork.gasPrice() + (s.configUints.Constants.CKEY_BEACON_FEE * 5);
2279 | }
2280 |
```


UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2273 | function estimateFee(uint256 _callbackGasLimit) public view returns (uint256 estimateFee) {  
2274 |     return  
2275 |     ((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +  
2276 |     _callbackGasLimit +  
2277 |     (s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1) * 3)) *  
2278 |     LibNetwork._gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);  
2279 | }
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2273 | function estimateFee(uint256 _callbackGasLimit) public view returns (uint256 estimateFee) {  
2274 |     return  
2275 |     ((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +  
2276 |     _callbackGasLimit +  
2277 |     (s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1) * 3)) *  
2278 |     LibNetwork._gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2275 | ((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +  
2276 | _callbackGasLimit +  
2277 | (s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1) * 3)) *  
2278 | LibNetwork._gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);  
2279 | }
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2275 | ((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +  
2276 | _callbackGasLimit +  
2277 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1) * 3)) *  
2278 | LibNetwork._gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);  
2279 | }
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2275 | ((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +  
2276 | _callbackGasLimit +  
2277 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1) * 3)) *  
2278 | LibNetwork._gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);  
2279 | }
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2276 | _callbackGasLimit +  
2277 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1) * 3)) *  
2278 | LibNetwork._gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);  
2279 | }  
2280 | }
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
2290 | {
2291 |     return
2292 |     (s_gasEstimates.Constants.GKEY_TOTAL_SUBMIT +
2293 |      _callbackGasLimit +
2294 |      (s_gasEstimates.Constants.GKEY_GAS_PER_BEACON_SELECT * (s_beacons.length - 1) * 3) *
2295 |      _gasPrice) + (s_configInts.Constants.CKEY_BEACON_FEE * 5);
2296 | }
2297 |
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
2290 | {
2291 |     return
2292 |     (s_gasEstimates.Constants.GKEY_TOTAL_SUBMIT +
2293 |      _callbackGasLimit +
2294 |      (s_gasEstimates.Constants.GKEY_GAS_PER_BEACON_SELECT * (s_beacons.length - 1) * 3) *
2295 |      _gasPrice) + (s_configInts.Constants.CKEY_BEACON_FEE * 5);
2296 | }
2297 |
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
Entry_flat.sol
Locations

```
2290 | {
2291 |     return
2292 |     ((s_gasEstimates.Constants.GKEY_TOTAL_SUBMIT +
2293 |      _callbackGasLimit +
2294 |      (s_gasEstimates.Constants.GKEY_GAS_PER_BEACON_SELECT * (s_beacons.length - 1) * 3) *
2295 |      _gasPrice) + (s_configInts.Constants.CKEY_BEACON_FEE * 5);
2296 | }
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2290 | {  
2291 |     return  
2292 |     (( $ gasEstimates.Constants.GKEY_TOTAL_SUBMIT | +  
2293 |     _callbackGasLimit | +  
2294 |     (( $ gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * ( $ beacons.length - 1 ) ) * 3 ) ) *  
2295 |     _gasPrice ) + ( $ configInts[Constants.CKEY_BEACON_FEE] * 5 );
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2292 | (( $ gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +  
2293 |     _callbackGasLimit +  
2294 |     ( $ gasEstimates.Constants.GKEY_GAS_PER_BEACON_SELECT * ( $ beacons.length - 1 ) ) * 3 ) ) *  
2295 |     _gasPrice ) + ( $ configInts[Constants.CKEY_BEACON_FEE] * 5 );  
2296 | }
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2292 | (( $ gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +  
2293 |     _callbackGasLimit +  
2294 |     ( $ gasEstimates.Constants.GKEY_GAS_PER_BEACON_SELECT * ( $ beacons.length - 1 ) ) * 3 ) ) *  
2295 |     _gasPrice ) + ( $ configInts[Constants.CKEY_BEACON_FEE] * 5 );  
2296 | }
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2292 | ((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
2293 | _callbackGasLimit +
2294 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
2295 | _gasPrice) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
2296 | }
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2293 | _callbackGasLimit +
2294 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
2295 | _gasPrice) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
2296 | }
2297 |
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2312 | if (
2313 | s.ethDeposit[msg.sender] < s.ethReserved[msg.sender] ||
2314 | _estimateFee > (s.ethDeposit[msg.sender] - s.ethReserved[msg.sender])
2315 | ) revert EthDepositTooLow(s.ethDeposit[msg.sender], s.ethReserved[msg.sender], _estimateFee);
2316 |
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2316 |  
2317 | // Increase the client's reserved ETH by the estimated fee  
2318 | s.ethReserved[msg.sender] += _estimateFee;  
2319 |  
2320 | // Increment the latest request ID and store it in the 'id' variable
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2319 |  
2320 | // Increment the latest request ID and store it in the 'id' variable  
2321 | s.latestRequestId++;  
2322 | id = s.latestRequestId;  
2323 |
```

UNKNOWN Arithmetic operation "+=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2470 | function beaconStakeEth(address _beacon) external payable {  
2471 | // Increase the beacon's ETH collateral by the value of the transaction  
2472 | s.ethCollateral[_beacon] += msg.value;  
2473 |  
2474 | // Emit an event to log the deposit of ETH by the beacon
```

UNKNOWN Arithmetic operation "-=" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2479 | function beaconUnstakeEth(uint256 _amount) external {
2480 |     // Decrease the beacon's ETH collateral by the specified amount
2481 |     s.ethCollateral[msg.sender] -= _amount;
2482 |
2483 |     // Check if the beacon's collateral is below the minimum required amount
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2698 | // If the consecutive submissions count is less than the maximum allowed, increment it
2699 | unchecked {
2700 |     memBeacon.consecutiveSubmissions++;
2701 | }
2702 | }
```

UNKNOWN Arithmetic operation "--" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2703 |
2704 | // Decrement the pending count for the beacon
2705 | if (memBeacon.pending > 0) memBeacon.pending--;
2706 |
2707 | // Save the updated Beacon struct
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2760 | if (msg.sender != s.sequencer) revert SenderNotBeaconOrSequencer();
2761 | // Calculate the earliest time that the sequencer can submit on behalf of the beacon
2762 | uint256 sequencerSubmitTime = data.timestamp + (data.expirationSeconds / 2);
2763 |
2764 | // Calculate the earliest block number that the sequencer can submit on behalf of the beacon
```

UNKNOWN Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2760 | if (msg.sender != s.sequencer) revert SenderNotBeaconOrSequencer();
2761 | // Calculate the earliest time that the sequencer can submit on behalf of the beacon
2762 | uint256 sequencerSubmitTime = data.timestamp + (data.expirationSeconds / 2);
2763 |
2764 | // Calculate the earliest block number that the sequencer can submit on behalf of the beacon
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2763 |
2764 | // Calculate the earliest block number that the sequencer can submit on behalf of the beacon
2765 | uint256 sequencerSubmitBlock = data.height + (data.expirationBlocks / 2);
2766 |
2767 | // Check if the sequencer is submitting too early
```


UNKNOWN Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2763 |  
2764 | // Calculate the earliest block number that the sequencer can submit on behalf of the beacon  
2765 | uint256 sequencerSubmitBlock = data.height + (data.expirationBlocks / 2);  
2766 |  
2767 | // Check if the sequencer is submitting too early
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
12 | address(this),  
13 | id,  
14 | blockhash(block.number - 1),  
15 | block.difficulty,  
16 | block.timestamp,
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1427 | // replace selector with last selector, then delete last selector  
1428 | uint256 selectorPosition = ds.selectorToFacetAndPosition[_selector].functionSelectorPosition;  
1429 | uint256 lastSelectorPosition = ds.facetFunctionSelectors[_facetAddress].functionSelectors.length - 1;  
1430 | // if not the same then replace _selector with lastSelector  
1431 | if (selectorPosition != lastSelectorPosition) {
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1443 | if (lastSelectorPosition == 0) {  
1444 | // replace facet address with last facet address and delete last facet address  
1445 | uint256 lastFacetAddressPosition = ds.facetAddresses.length - 1;  
1446 | uint256 facetAddressPosition = ds.facetFunctionSelectors[_facetAddress].facetAddressPosition;  
1447 | if (facetAddressPosition != lastFacetAddressPosition) {
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
1590 | uint256 index = s.beaconIndex[_beacon];  
1591 | if (index == 0) revert BeaconNotFound();  
1592 | uint256 lastBeaconIndex = s.beacons.length - 1;  
1593 | s.beacon[_beacon].registered = false;  
1594 | if (index == lastBeaconIndex) {
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2010 |  
2011 | // Checks if enough beacons are available to replace with  
2012 | if (s.beacons.length < 5 || beaconsToStrikeLen * 2 > s.beacons.length - 1)  
2013 | revert NotEnoughBeaconsAvailable(  
2014 | s.beacons.length,
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2142 | address(this),
2143 | _request,
2144 | LibNetwork._blockHash(LibNetwork._blockNumber() - 1),
2145 | block.chainid
2146 | )
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2275 | ((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
2276 | _callbackGasLimit +
2277 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
2278 | LibNetwork._gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
2279 | }
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

Entry_flat.sol

Locations

```
2292 | ((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
2293 | _callbackGasLimit +
2294 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
2295 | _gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
2296 | }
```

LOW

A floating pragma is set.

SWC-103

The current pragma Solidity directive is ""^0.8.17"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

Entry_flat.sol

Locations

```
1 |
2 | // File: contracts/libraries/LibNetwork.sol
3 | pragma solidity ^0.8.17
4 |
5 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
270 | uint256[4] memory zs;
271 | // z1^2, z1^3, z2^2, z2^3
272 | zs[0] = mulmod(_z1, _z1, PP);
273 | zs[1] = mulmod(_z1, zs[0], PP);
274 | zs[2] = mulmod(_z2, _z2, PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
271 | // z1^2, z1^3, z2^2, z2^3
272 | zs[0] = mulmod(_z1, _z1, PP);
273 | zs[1] = mulmod(_z1, zs[0], PP);
274 | zs[2] = mulmod(_z2, _z2, PP);
275 | zs[3] = mulmod(_z2, zs[2], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
271 | // z1^2, z1^3, z2^2, z2^3
272 | zs[0] = mulmod(_z1, _z1, PP);
273 | zs[1] = mulmod(_z1, zs[0], PP);
274 | zs[2] = mulmod(_z2, _z2, PP);
275 | zs[3] = mulmod(_z2, zs[2], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
272 | zs[0] = mulmod(_z1, _z1, PP);
273 | zs[1] = mulmod(_z1, zs[0], PP);
274 | zs[2] = mulmod(_z2, _z2, PP);
275 | zs[3] = mulmod(_z2, zs[2], PP);
276 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
273 | zs[1] = mulmod(_z1, zs[0], PP);
274 | zs[2] = mulmod(_z2, _z2, PP);
275 | zs[3] = mulmod(_z2, zs[2], PP);
276 |
277 | // u1, s1, u2, s2
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
273 | zs[1] = mulmod(_z1, zs[0], PP);
274 | zs[2] = mulmod(_z2, _z2, PP);
275 | zs[3] = mulmod(_z2, zs[2], PP);
276 |
277 | // u1, s1, u2, s2
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
276 |
277 | // u1, s1, u2, s2
278 | zs = [mulmod(_x1, zs[2], PP), mulmod(_y1, zs[3], PP), mulmod(_x2, zs[0], PP), mulmod(_y2, zs[1], PP)];
279 |
280 | // In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
276 |
277 | // u1, s1, u2, s2
278 | zs = [mulmod(_x1, zs[2], PP), mulmod(_y1, zs[3], PP), mulmod(_x2, zs[0], PP), mulmod(_y2, zs[1], PP)];
279 |
280 | // In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
276 |  
277 | // u1, s1, u2, s2  
278 | zs = [mulmod(_x1, zs[2], PP), mulmod(_y1, zs[3], PP), mulmod(_x2, zs[0], PP), mulmod(_y2, zs[1], PP)];  
279 |  
280 | // In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
276 |  
277 | // u1, s1, u2, s2  
278 | zs = [mulmod(_x1, zs[2], PP), mulmod(_y1, zs[3], PP), mulmod(_x2, zs[0], PP), mulmod(_y2, zs[1], PP)];  
279 |  
280 | // In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
279 |  
280 | // In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used  
281 | require(zs[0] != zs[2] || zs[1] != zs[3], "Use jacDouble function instead");  
282 |  
283 | uint256[4] memory hr;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
279 |  
280 | // In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used  
281 | require(zs[0] != zs[2] || zs[1] != zs[3], "Use jacDouble function instead");  
282 |  
283 | uint256[4] memory hr;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
279 |  
280 | // In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used  
281 | require(zs[0] != zs[2] || zs[1] != zs[3], "Use jacDouble function instead");  
282 |  
283 | uint256[4] memory hr;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
279 |  
280 | // In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used  
281 | require(zs[0] != zs[2] || zs[1] != zs[3], "Use jacDouble function instead");  
282 |  
283 | uint256[4] memory hr;
```


UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
283 | uint256[4] memory hr;  
284 | //h  
285 | hr[0] = addmod(zs[2], PP - zs[0], PP);  
286 | //r  
287 | hr[1] = addmod(zs[3], PP - zs[1], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
283 | uint256[4] memory hr;  
284 | //h  
285 | hr[0] = addmod(zs[2], PP - zs[0], PP);  
286 | //r  
287 | hr[1] = addmod(zs[3], PP - zs[1], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
283 | uint256[4] memory hr;  
284 | //h  
285 | hr[0] = addmod(zs[2], PP - zs[0], PP);  
286 | //r  
287 | hr[1] = addmod(zs[3], PP - zs[1], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
285 | hr[0] = addmod(zs[2], PP - zs[0], PP);
286 | //r
287 | hr[1] = addmod(zs[3], PP - zs[1], PP);
288 | //h^2
289 | hr[2] = mulmod(hr[0], hr[0], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
285 | hr[0] = addmod(zs[2], PP - zs[0], PP);
286 | //r
287 | hr[1] = addmod(zs[3], PP - zs[1], PP);
288 | //h^2
289 | hr[2] = mulmod(hr[0], hr[0], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
285 | hr[0] = addmod(zs[2], PP - zs[0], PP);
286 | //r
287 | hr[1] = addmod(zs[3], PP - zs[1], PP);
288 | //h^2
289 | hr[2] = mulmod(hr[0], hr[0], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
287 | hr[1] = addmod(zs[3], PP - zs[1], PP);
288 | //h^2
289 | hr[2] = mulmod(hr[0], hr[0], PP);
290 | // h^3
291 | hr[3] = mulmod(hr[2], hr[0], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
287 | hr[1] = addmod(zs[3], PP - zs[1], PP);
288 | //h^2
289 | hr[2] = mulmod(hr[0], hr[0], PP);
290 | // h^3
291 | hr[3] = mulmod(hr[2], hr[0], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
287 | hr[1] = addmod(zs[3], PP - zs[1], PP);
288 | //h^2
289 | hr[2] = mulmod(hr[0], hr[0], PP);
290 | // h^3
291 | hr[3] = mulmod(hr[2], hr[0], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
289 | hr[2] = mulmod(hr[0], hr[0], PP);  
290 | // h^3  
291 | hr[3] = mulmod(hr[2], hr[0], PP);  
292 | // qx = -h^3 -2u1h^2+r^2  
293 | uint256 qx = addmod(mulmod(hr[1], hr[1], PP), PP - hr[3], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
289 | hr[2] = mulmod(hr[0], hr[0], PP);  
290 | // h^3  
291 | hr[3] = mulmod(hr[2], hr[0], PP);  
292 | // qx = -h^3 -2u1h^2+r^2  
293 | uint256 qx = addmod(mulmod(hr[1], hr[1], PP), PP - hr[3], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
289 | hr[2] = mulmod(hr[0], hr[0], PP);  
290 | // h^3  
291 | hr[3] = mulmod(hr[2], hr[0], PP);  
292 | // qx = -h^3 -2u1h^2+r^2  
293 | uint256 qx = addmod(mulmod(hr[1], hr[1], PP), PP - hr[3], PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
291 | hr[3] = mulmod(hr[2], hr[0], PP);
292 | // qx = -h^3 -2u1h^2+r^2
293 | uint256 qx = addmod(mulmod(hr[1], hr[1], PP), PP - hr[3], PP);
294 | qx = addmod(qx, PP - mulmod(2, mulmod(zs[0], hr[2], PP), PP), PP);
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
291 | hr[3] = mulmod(hr[2], hr[0], PP);
292 | // qx = -h^3 -2u1h^2+r^2
293 | uint256 qx = addmod(mulmod(hr[1], hr[1], PP), PP - hr[3], PP);
294 | qx = addmod(qx, PP - mulmod(2, mulmod(zs[0], hr[2], PP), PP), PP);
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
291 | hr[3] = mulmod(hr[2], hr[0], PP);
292 | // qx = -h^3 -2u1h^2+r^2
293 | uint256 qx = addmod(mulmod(hr[1], hr[1], PP), PP - hr[3], PP);
294 | qx = addmod(qx, PP - mulmod(2, mulmod(zs[0], hr[2], PP), PP), PP);
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
292 | // qx = -h^3 -2u1h^2+r^2
293 | uint256 qx = addmod(mulmod(hr[1], hr[1], PP), PP - hr[3], PP);
294 | qx = addmod(qx, PP - mulmod(2, mulmod(zs[0], hr[2], PP), PP), PP);
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
296 | uint256 qy = mulmod(hr[1], addmod(mulmod(zs[0], hr[2], PP), PP - qx, PP), PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
292 | // qx = -h^3 -2u1h^2+r^2
293 | uint256 qx = addmod(mulmod(hr[1], hr[1], PP), PP - hr[3], PP);
294 | qx = addmod(qx, PP - mulmod(2, mulmod(zs[0], hr[2], PP), PP), PP);
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
296 | uint256 qy = mulmod(hr[1], addmod(mulmod(zs[0], hr[2], PP), PP - qx, PP), PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
294 | qx = addmod(qx, PP - mulmod(2, mulmod(zs[0], hr[2], PP), PP), PP);
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
296 | uint256 qy = mulmod(hr[1], addmod(mulmod(zs[0], hr[2], PP), PP - qx, PP), PP);
297 | qy = addmod(qy, PP - mulmod(zs[1], hr[3], PP), PP);
298 | // qz = h*z1*z2
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
294 | qx = addmod(qx, PP - mulmod(2, mulmod(zs[0], hr[2], PP), PP), PP);
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
296 | uint256 qy = mulmod(hr[1], addmod(mulmod(zs[0], hr[2], PP), PP - qx, PP), PP);
297 | qy = addmod(qy, PP - mulmod(zs[1], hr[3], PP), PP);
298 | // qz = h*z1*z2
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
294 | qx = addmod(qx, PP - mulmod(2, mulmod(zs[0], hr[2], PP), PP), PP);
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
296 | uint256 qy = mulmod(hr[1], addmod(mulmod(zs[0], hr[2], PP), PP - qx, PP), PP);
297 | qy = addmod(qy, PP - mulmod(zs[1], hr[3], PP), PP);
298 | // qz = h*z1*z2
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
296 | uint256 qy = mulmod(hr[1], addmod(mulmod(zs[0], hr[2], PP), PP - qx, PP), PP);
297 | qy = addmod(qy, PP - mulmod(zs[1], hr[3], PP), PP);
298 | // qz = h*z1*z2
299 | uint256 qz = mulmod(hr[0], mulmod(_z1, _z2, PP), PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
295 | // qy = -s1*z1*h^3+r(u1*h^2 -x^3)
296 | uint256 qy = mulmod(hr[1], addmod(mulmod(zs[0], hr[2], PP), PP - qx, PP), PP);
297 | qy = addmod(qy, PP - mulmod(zs[1], hr[3], PP), PP);
298 | // qz = h*z1*z2
299 | uint256 qz = mulmod(hr[0], mulmod(_z1, _z2, PP), PP);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
297 | qy = addmod(qy, PP - mulmod(zs[1], hr[3], PP), PP);
298 | // qz = h*z1*z2
299 | uint256 qz = mulmod(hr[0], mulmod(_z1, _z2, PP), PP);
300 | return (qx, qy, qz);
301 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
472 | // Step 3: U = s*B - c*Y (where B is the generator)
473 | (uint256 uPointX, uint256 uPointY) = ecMulSubMul(
474 | _proof[3],
475 | GX,
476 | GY,
```


UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
475 | GX,  
476 | GY,  
477 | _proof[2],  
478 | _publicKey[0],  
479 | _publicKey[1]
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
476 | GY,  
477 | _proof[2],  
478 | _publicKey[0],  
479 | _publicKey[1]  
480 | );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
477 | _proof[2],  
478 | _publicKey[0],  
479 | _publicKey[1]  
480 | );  
481 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
482 | // Step 4: V = s*H - c*Gamma
483 | (uint256 vPointX, uint256 vPointY) = ecMulSubMul(
484 |   _proof[3],
485 |   hPointX,
486 |   hPointY,
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
485 | hPointX,
486 | hPointY,
487 |   _proof[2],
488 |   _proof[0],
489 |   _proof[1]
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
486 | hPointY,
487 |   _proof[2],
488 |   _proof[0],
489 |   _proof[1]
490 | );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
487 | _proof[2],  
488 | _proof[0],  
489 | _proof[1]  
490 | );  
491 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
494 | hPointX,  
495 | hPointY,  
496 | _proof[0],  
497 | _proof[1],  
498 | uPointX,
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
495 | hPointY,  
496 | _proof[0],  
497 | _proof[1],  
498 | uPointX,  
499 | uPointY,
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
503 |
504 | // Step 6: Check validity c == c'
505 | return uint128(derivedC) == _proof[2];
506 | }
507 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
530 | if (
531 |   !ecMulSubMulVerify(
532 |     _proof[3], //s
533 |     _proof[2], //c
534 |     _publicKey[0], //Y-x
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
531 | !ecMulSubMulVerify(
532 |   _proof[3], //s
533 |   _proof[2], //c
534 |   _publicKey[0], //Y-x
535 |   _publicKey[1], //Y-y
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
532 | _proof[3], //s
533 | _proof[2], //c
534 | _publicKey[0], //Y-x
535 | _publicKey[1], //Y-y
536 | _uPoint[0], //U-x
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
533 | _proof[2], //c
534 | _publicKey[0], //Y-x
535 | _publicKey[1], //Y-y
536 | _uPoint[0], //U-x
537 | _uPoint[1]
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
534 | _publicKey[0], //Y-x
535 | _publicKey[1], //Y-y
536 | _uPoint[0], //U-x
537 | _uPoint[1]
538 | ) || //U-y
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
535 | _publicKey[1], //Y-y
536 | _uPoint[0], //U-x
537 | _uPoint[1]
538 | ) || //U-y
539 | !ecMulVerify(
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
538 | ) || //U-y
539 | !ecMulVerify(
540 | _proof[3], //s
541 | hPointX, //H-x
542 | hPointY, //H-y
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
541 | hPointX, //H-x
542 | hPointY, //H-y
543 | _vComponents[0], //s*H -x
544 | _vComponents[1]
545 | ) || //s*H -y
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
542 | hPointY, //H-y
543 | _vComponents[0], //s*H -x
544 | _vComponents[1]
545 | ) || //s*H -y
546 | !ecMulVerify(
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
545 | ) || //s*H -y
546 | !ecMulVerify(
547 | _proof[2], //c
548 | _proof[0], //gamma-x
549 | _proof[1], //gamma-y
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
546 | !ecMulVerify(
547 | _proof[2], //c
548 | _proof[0], //gamma-x
549 | _proof[1], //gamma-y
550 | _vComponents[2], //c*Gamma -x
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
547 | _proof[2], //c
548 | _proof[0], //gamma-x
549 | _proof[1], //gamma-y
550 | _vComponents[2], //c*Gamma -x
551 | _vComponents[3]
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
548 | _proof[0], //gamma-x
549 | _proof[1], //gamma-y
550 | _vComponents[2], //c*Gamma -x
551 | _vComponents[3]
552 | ) //c*Gamma -y
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
549 | _proof[1], //gamma-y
550 | _vComponents[2], //c*Gamma -x
551 | _vComponents[3]
552 | ) //c*Gamma -y
553 | {
```


UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
556 |
557 | (uint256 vPointX, uint256 vPointY) = EllipticCurve.ecSub(
558 |   _vComponents[0], //s*H -x
559 |   _vComponents[1], //s*H -y
560 |   _vComponents[2], //c*Gamma -x
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
557 | (uint256 vPointX, uint256 vPointY) = EllipticCurve.ecSub(
558 |   _vComponents[0], //s*H -x
559 |   _vComponents[1], //s*H -y
560 |   _vComponents[2], //c*Gamma -x
561 |   _vComponents[3] //c*Gamma -y
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
558 |   _vComponents[0], //s*H -x
559 |   _vComponents[1], //s*H -y
560 |   _vComponents[2], //c*Gamma -x
561 |   _vComponents[3] //c*Gamma -y
562 | );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
559 | _vComponents[1], //s*H -y
560 | _vComponents[2], //c*Gamma -x
561 | _vComponents[3] //c*Gamma -y
562 | );
563 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
566 | hPointX,
567 | hPointY,
568 | _proof[0],
569 | _proof[1],
570 | _uPoint[0],
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
567 | hPointY,
568 | _proof[0],
569 | _proof[1],
570 | _uPoint[0],
571 | _uPoint[1],
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
568 | _proof[0],  
569 | _proof[1],  
570 | _uPoint[0],  
571 | _uPoint[1],  
572 | vPointX,
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
569 | _proof[1],  
570 | _uPoint[0],  
571 | _uPoint[1],  
572 | vPointX,  
573 | vPointY
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
575 |  
576 | // Step 6: Check validity c == c'  
577 | return uint128(derivedC) == _proof[2];  
578 | }  
579 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
627 | (uint256 hPointX, uint256 hPointY) = hashToTryAndIncrement(_publicKey, _message);
628 | (uint256 uPointX, uint256 uPointY) = ecMulSubMul(
629 |   _proof[3],
630 |   GX,
631 |   GY,
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
630 | GX,
631 | GY,
632 |   _proof[2],
633 |   _publicKey[0],
634 |   _publicKey[1]
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
631 | GY,
632 |   _proof[2],
633 |   _publicKey[0],
634 |   _publicKey[1]
635 | );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
632 | _proof[2],
633 | _publicKey[0],
634 | _publicKey[1]
635 | );
636 | // Requirements for Step 4: V = s*H - c*Gamma
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
635 | );
636 | // Requirements for Step 4: V = s*H - c*Gamma
637 | (uint256 sHX, uint256 sHY) = derivePoint(_proof[3], hPointX, hPointY);
638 | (uint256 cGammaX, uint256 cGammaY) = derivePoint(_proof[2], _proof[0], _proof[1]);
639 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
636 | // Requirements for Step 4: V = s*H - c*Gamma
637 | (uint256 sHX, uint256 sHY) = derivePoint(_proof[3], hPointX, hPointY);
638 | (uint256 cGammaX, uint256 cGammaY) = derivePoint(_proof[2], _proof[0], _proof[1]);
639 |
640 | return ([uPointX, uPointY], [sHX, sHY, cGammaX, cGammaY]);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
636 // Requirements for Step 4: V = s*H - c*Gamma
637 (uint256 sHX, uint256 sHY) = derivePoint(_proof[3], hPointX, hPointY);
638 (uint256 cGammaX, uint256 cGammaY) = derivePoint(_proof[2], _proof[0], _proof[1]);
639
640 return ([uPointX, uPointY], [sHX, sHY, cGammaX, cGammaY]);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
636 // Requirements for Step 4: V = s*H - c*Gamma
637 (uint256 sHX, uint256 sHY) = derivePoint(_proof[3], hPointX, hPointY);
638 (uint256 cGammaX, uint256 cGammaY) = derivePoint(_proof[2], _proof[0], _proof[1]);
639
640 return ([uPointX, uPointY], [sHX, sHY, cGammaX, cGammaY]);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
659 uint8(1),
660 // Public Key
661 encodePoint(_publicKey[0], _publicKey[1]),
662 // Message
663 _message
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
659 | uint8(1),
660 | // Public Key
661 | encodePoint(_publicKey[0], _publicKey[1]),
662 | // Message
663 | _message
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1046 | /// @return SAccounts struct
1047 | function _resolveAddressCalldata(address[4] calldata _data) internal pure returns (SAccounts memory) {
1048 |     return SAccounts(_data[0], [_data[1], _data[2], _data[3]]);
1049 | }
1050 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1046 | /// @return SAccounts struct
1047 | function _resolveAddressCalldata(address[4] calldata _data) internal pure returns (SAccounts memory) {
1048 |     return SAccounts(_data[0], [_data[1], _data[2], _data[3]]);
1049 | }
1050 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1046 | /// @return SAccounts struct
1047 | function _resolveAddressCalldata(address[4] calldata _data) internal pure returns (SAccounts memory) {
1048 |     return SAccounts(_data[0], [_data[1], _data[2], _data[3]]);
1049 | }
1050 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1046 | /// @return SAccounts struct
1047 | function _resolveAddressCalldata(address[4] calldata _data) internal pure returns (SAccounts memory) {
1048 |     return SAccounts(_data[0], [_data[1], _data[2], _data[3]]);
1049 | }
1050 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1059 | return
1060 | SPackedSubmitData(
1061 |     uint256(_data[0]),
1062 |     SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),
1063 |     SFastVerifyData(
```


UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1060 | SPackedSubmitData(  
1061 | uint256(_data[0]),  
1062 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),  
1063 | SFastVerifyData(  
1064 | [_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1060 | SPackedSubmitData(  
1061 | uint256(_data[0]),  
1062 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),  
1063 | SFastVerifyData(  
1064 | [_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1060 | SPackedSubmitData(  
1061 | uint256(_data[0]),  
1062 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),  
1063 | SFastVerifyData(  
1064 | [_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1060 | SPackedSubmitData(  
1061 | uint256(_data[0]),  
1062 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),  
1063 | SFastVerifyData(  
1064 | [_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1060 | SPackedSubmitData(  
1061 | uint256(_data[0]),  
1062 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),  
1063 | SFastVerifyData(  
1064 | [_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1060 | SPackedSubmitData(  
1061 | uint256(_data[0]),  
1062 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),  
1063 | SFastVerifyData(  
1064 | [_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1060 | SPackedSubmitData(  
1061 |   uint256(_data[0]),  
1062 |   SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),  
1063 |   SFastVerifyData(  
1064 |     [_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1062 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),  
1063 | SFastVerifyData(  
1064 |   [_data[8], _data[9], _data[10], _data[11]],  
1065 |   [_data[12], _data[13]],  
1066 |   [_data[14], _data[15], _data[16], _data[17]]
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1062 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),  
1063 | SFastVerifyData(  
1064 |   [_data[8], _data[9], _data[10], _data[11]],  
1065 |   [_data[12], _data[13]],  
1066 |   [_data[14], _data[15], _data[16], _data[17]]
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1062 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),
1063 | SFastVerifyData(
1064 | [_data[8], _data[9], _data[10], _data[11]],
1065 | [_data[12], _data[13]],
1066 | [_data[14], _data[15], _data[16], _data[17]]
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1062 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),
1063 | SFastVerifyData(
1064 | [_data[8], _data[9], _data[10], _data[11]],
1065 | [_data[12], _data[13]],
1066 | [_data[14], _data[15], _data[16], _data[17]]
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1063 | SFastVerifyData(
1064 | [_data[8], _data[9], _data[10], _data[11]],
1065 | [_data[12], _data[13]],
1066 | [_data[14], _data[15], _data[16], _data[17]]
1067 | )
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1063 | SFastVerifyData(  
1064 |   [_data[8], _data[9], _data[10], _data[11]],  
1065 |   [_data[12], _data[13]],  
1066 |   [_data[14], _data[15], _data[16], _data[17]]  
1067 | )
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1064 |   [_data[8], _data[9], _data[10], _data[11]],  
1065 |   [_data[12], _data[13]],  
1066 |   [_data[14], _data[15], _data[16], _data[17]]  
1067 | )  
1068 | );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1064 |   [_data[8], _data[9], _data[10], _data[11]],  
1065 |   [_data[12], _data[13]],  
1066 |   [_data[14], _data[15], _data[16], _data[17]]  
1067 | )  
1068 | );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1064 | [_data[8], _data[9], _data[10], _data[11]],  
1065 | [_data[12], _data[13]],  
1066 | [_data[14], _data[15], _data[16], _data[17]]  
1067 | )  
1068 | );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1064 | [_data[8], _data[9], _data[10], _data[11]],  
1065 | [_data[12], _data[13]],  
1066 | [_data[14], _data[15], _data[16], _data[17]]  
1067 | )  
1068 | );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1075 | return  
1076 | SPackedUintData(  
1077 | uint256(_data[0]),  
1078 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])  
1079 | );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1076 | SPackedUintData(  
1077 | uint256(_data[0]),  
1078 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])  
1079 | );  
1080 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1076 | SPackedUintData(  
1077 | uint256(_data[0]),  
1078 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])  
1079 | );  
1080 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1076 | SPackedUintData(  
1077 | uint256(_data[0]),  
1078 | SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])  
1079 | );  
1080 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1076 | SPackedUintData(  
1077 |   uint256(_data[0]),  
1078 |   SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])  
1079 | );  
1080 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1076 | SPackedUintData(  
1077 |   uint256(_data[0]),  
1078 |   SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])  
1079 | );  
1080 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1076 | SPackedUintData(  
1077 |   uint256(_data[0]),  
1078 |   SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])  
1079 | );  
1080 | }
```


UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1076 | SPackedUintData(  
1077 |   uint256(_data[0]),  
1078 |   SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])  
1079 | );  
1080 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1327 | } internal {  
1328 |   for (uint256 facetIndex; facetIndex < _diamondCut.length; facetIndex++) {  
1329 |     IDiamondCut.FacetCutAction action = _diamondCut[facetIndex].action;  
1330 |     if (action == IDiamondCut.FacetCutAction.Add) {  
1331 |       addFunctions(_diamondCut[facetIndex].facetAddress, _diamondCut[facetIndex].functionSelectors);  
1332 |     } else if (action == IDiamondCut.FacetCutAction.Replace) {  
1333 |       replaceFunctions(  
1334 |         _diamondCut[facetIndex].facetAddress, _diamondCut[facetIndex].functionSelectors,  
1335 |         _diamondCut[facetIndex].functionSelectors,  
1336 |         _diamondCut[facetIndex].functionSelectors);  
1337 |     }  
1338 |   }  
1339 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1329 | IDiamondCut.FacetCutAction action = _diamondCut[facetIndex].action;  
1330 | if (action == IDiamondCut.FacetCutAction.Add) {  
1331 |   addFunctions(_diamondCut[facetIndex].facetAddress, _diamondCut[facetIndex].functionSelectors);  
1332 | } else if (action == IDiamondCut.FacetCutAction.Replace) {  
1333 |   replaceFunctions(  
1334 |     _diamondCut[facetIndex].facetAddress, _diamondCut[facetIndex].functionSelectors,  
1335 |     _diamondCut[facetIndex].functionSelectors,  
1336 |     _diamondCut[facetIndex].functionSelectors);  
1337 | }  
1338 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1329 | IDiamondCut.FacetCutAction action = _diamondCut[facetIndex].action;
1330 | if (action == IDiamondCut.FacetCutAction.Add) {
1331 |     addFunctions(_diamondCut[facetIndex].facetAddress, _diamondCut[facetIndex].functionSelectors);
1332 | } else if (action == IDiamondCut.FacetCutAction.Replace) {
1333 |     replaceFunctions(
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1332 | } else if (action == IDiamondCut.FacetCutAction.Replace) {
1333 |     replaceFunctions(
1334 |         _diamondCut[facetIndex].facetAddress,
1335 |         _diamondCut[facetIndex].functionSelectors
1336 |     );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1333 | replaceFunctions(
1334 |     _diamondCut[facetIndex].facetAddress,
1335 |     _diamondCut[facetIndex].functionSelectors
1336 | );
1337 | else if (action == IDiamondCut.FacetCutAction.Remove) {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1337 | } else if (action == IDiamondCut.FacetCutAction.Remove) {  
1338 |     removeFunctions(  
1339 |         _diamondCut[facetIndex].facetAddress,  
1340 |         _diamondCut[facetIndex].functionSelectors  
1341 |     );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1338 |     removeFunctions(  
1339 |         _diamondCut[facetIndex].facetAddress,  
1340 |         _diamondCut[facetIndex].functionSelectors  
1341 |     );  
1342 | } else {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1358 | }  
1359 | for (uint256 selectorIndex; selectorIndex < _functionSelectors.length; selectorIndex++) {  
1360 |     bytes4 selector = _functionSelectors[selectorIndex];  
1361 |     address oldFacetAddress = ds.selectorToFacetAndPosition[selector].facetAddress;  
1362 |     require(oldFacetAddress == address(0), "LibDiamondCut: Can't add function that already exists");
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1376 | }
1377 | for (uint256 selectorIndex; selectorIndex < _functionSelectors.length; selectorIndex++) {
1378 |     bytes4 selector = _functionSelectors[selectorIndex];
1379 |     address oldFacetAddress = ds.selectorToFacetAndPosition[selector].facetAddress;
1380 |     require(
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1394 | require(_facetAddress == address(0), "LibDiamondCut: Remove facet address must be address(0)");
1395 | for (uint256 selectorIndex; selectorIndex < _functionSelectors.length; selectorIndex++) {
1396 |     bytes4 selector = _functionSelectors[selectorIndex];
1397 |     address oldFacetAddress = ds.selectorToFacetAndPosition[selector].facetAddress;
1398 |     removeFunction(ds, oldFacetAddress, selector);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1430 | // if not the same then replace _selector with lastSelector
1431 | if (selectorPosition != lastSelectorPosition) {
1432 |     bytes4 lastSelector = ds.facetFunctionSelectors[_facetAddress].functionSelectors[
1433 |         lastSelectorPosition
1434 |     ];
1435 |     ds.facetFunctionSelectors[_facetAddress].functionSelectors[selectorPosition] = lastSelector;
1436 |     ds.selectorToFacetAndPosition[lastSelector].functionSelectorPosition = uint96(selectorPosition);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1433 | lastSelectorPosition
1434 | ];
1435 | ds.facetFunctionSelectors[_facetAddress].functionSelectors[selectorPosition] = lastSelector;
1436 | ds.selectorToFacetAndPosition[lastSelector].functionSelectorPosition = uint96(selectorPosition);
1437 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1446 | uint256 facetAddressPosition = ds.facetFunctionSelectors[_facetAddress].facetAddressPosition;
1447 | if (facetAddressPosition != lastFacetAddressPosition) {
1448 |     address lastFacetAddress = ds.facetAddresses[lastFacetAddressPosition];
1449 |     ds.facetAddresses[facetAddressPosition] = lastFacetAddress;
1450 |     ds.facetFunctionSelectors[lastFacetAddress].facetAddressPosition = facetAddressPosition;
1451 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1447 | if (facetAddressPosition != lastFacetAddressPosition) {
1448 |     address lastFacetAddress = ds.facetAddresses[lastFacetAddressPosition];
1449 |     ds.facetAddresses[facetAddressPosition] = lastFacetAddress;
1450 |     ds.facetFunctionSelectors[lastFacetAddress].facetAddressPosition = facetAddressPosition;
1451 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1597 | return;  
1598 | }  
1599 | s.beacons[index] = s.beacons[lastBeaconIndex];  
1600 | address newBeacon = s.beacons[lastBeaconIndex];  
1601 | s.beaconIndex[_beacon] = 0;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1597 | return;  
1598 | }  
1599 | s.beacons[index] = s.beacons.lastBeaconIndex;  
1600 | address newBeacon = s.beacons[lastBeaconIndex];  
1601 | s.beaconIndex[_beacon] = 0;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1598 | }  
1599 | s.beacons[index] = s.beacons[lastBeaconIndex];  
1600 | address newBeacon = s.beacons.lastBeaconIndex;  
1601 | s.beaconIndex[_beacon] = 0;  
1602 | // The replacing beacon gets assigned the replaced beacon's index
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1615 | _data.height = LibNetwork._blockNumber();
1616 | _data.timestamp = block.timestamp;
1617 | address randomBeacon = _selectOneBeacon(_seed, [_accounts.beacons[0], _accounts.beacons[1]]);
1618 | s.beacon[randomBeacon].pending++;
1619 | _accounts.beacons[_beaconPos] = randomBeacon;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1615 | _data.height = LibNetwork._blockNumber();
1616 | _data.timestamp = block.timestamp;
1617 | address randomBeacon = _selectOneBeacon(_seed, [_accounts.beacons[0], _accounts.beacons[1]]);
1618 | s.beacon[randomBeacon].pending++;
1619 | _accounts.beacons[_beaconPos] = randomBeacon;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1617 | address randomBeacon = _selectOneBeacon(_seed, [_accounts.beacons[0], _accounts.beacons[1]]);
1618 | s.beacon[randomBeacon].pending++;
1619 | _accounts.beacons[_beaconPos] = randomBeacon;
1620 | s.requestToHash[_id] = LibBeacon._generateRequestHash(_id, _accounts, _data, _seed);
1621 | emit Events.RequestBeacon(_id, randomBeacon, _seed, _data.timestamp);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1632 | uint256 j = (uint256(keccak256(abi.encodePacked(_random, i))) % (selectedItems.length - i)) + i;
1633 | // Swap the items at indices i and j
1634 | address temp = selectedItems[i];
1635 | selectedItems[i] = selectedItems[j];
1636 | selectedItems[j] = temp;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1633 | // Swap the items at indices i and j
1634 | address temp = selectedItems[i];
1635 | selectedItems[i] = selectedItems[j];
1636 | selectedItems[j] = temp;
1637 | s.beacon[selectedItems[i]].pending++;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1633 | // Swap the items at indices i and j
1634 | address temp = selectedItems[i];
1635 | selectedItems[i] = selectedItems[j];
1636 | selectedItems[j] = temp;
1637 | s.beacon[selectedItems[i]].pending++;
```


UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1634 | address temp = selectedItems[i];
1635 | selectedItems[i] = selectedItems[j];
1636 | selectedItems[j] = temp;
1637 | s.beacon[selectedItems[i]].pending++;
1638 | unchecked {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1635 | selectedItems[i] = selectedItems[j];
1636 | selectedItems[j] = temp;
1637 | s.beacon[selectedItems[i]].pending++;
1638 | unchecked {
1639 | ++i;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1641 | } while (i < 3);
1642 | // Return the first two items from the shuffled array
1643 | return (selectedItems[1], selectedItems[2]);
1644 | }
1645 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1641 | } while (i < 3);
1642 | // Return the first two items from the shuffled array
1643 | return (selectedItems[1], selectedItems[2]);
1644 | }
1645 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1651 | uint256 j = uint256(_random) % count;
1652 |
1653 | return selectedItems[j];
1654 | }
1655 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1672 | uint256 j = 0;
1673 | while (j < _excluded.length) {
1674 | if (s.beacons[i] == _excluded[j]) {
1675 | found = true;
1676 | break;

```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1672 | uint256 j = 0;  
1673 | while (j < _excluded.length) {  
1674 |   if (s.beacons[i] == _excluded[j]) {  
1675 |     found = true;  
1676 |     break;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1681 | }  
1682 | if (!found) {  
1683 |   selectedItems[count] = s.beacons[i];  
1684 |   unchecked {  
1685 |     ++count;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1681 | }  
1682 | if (!found) {  
1683 |   selectedItems[count] = s.beacons[i];  
1684 |   unchecked {  
1685 |     ++count;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1707 | uint256 j = 0;
1708 | while (j < _excluded.length) {
1709 |   if (s.beacons[i] == _excluded[j]) {
1710 |     found = true;
1711 |     break;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1707 | uint256 j = 0;
1708 | while (j < _excluded.length) {
1709 |   if (s.beacons[i] == _excluded[j]) {
1710 |     found = true;
1711 |     break;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1716 | }
1717 | if (!found) {
1718 |   selectedItems.count = s.beacons[i];
1719 |   unchecked {
1720 |     ++count;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1716 | }  
1717 | if (!found) {  
1718 |   selectedItems[count] = s.beacons[i];  
1719 |   unchecked {  
1720 |     ++count;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1742 | uint256 j = 0;  
1743 | while (j < _excluded.length) {  
1744 |   if (s.beacons[i] == _excluded[j]) {  
1745 |     found = true;  
1746 |     break;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1742 | uint256 j = 0;  
1743 | while (j < _excluded.length) {  
1744 |   if (s.beacons[i] == _excluded[j]) {  
1745 |     found = true;  
1746 |     break;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1751 | }
1752 | if (!found) {
1753 |   selectedItems[count] = s.beacons[i];
1754 |   unchecked {
1755 |     ++count;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1751 | }
1752 | if (!found) {
1753 |   selectedItems[count] = s.beacons[i];
1754 |   unchecked {
1755 |     ++count;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1771 | uint256 _ethReserved
1772 | ) internal {
1773 |   bytes32 result = keccak256(abi.encodePacked(hashes[0], hashes[1], hashes[2]));
1774 |
1775 |   // Callback to requesting contract
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1771 | uint256 _ethReserved
1772 | ) internal {
1773 | bytes32 result = keccak256(abi.encodePacked(hashes[0], hashes[1], hashes[2]));
1774 |
1775 | // Callback to requesting contract
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1771 | uint256 _ethReserved
1772 | ) internal {
1773 | bytes32 result = keccak256(abi.encodePacked(hashes[0], hashes[1], hashes[2]));
1774 |
1775 | // Callback to requesting contract
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1942 | ) external {
1943 | // 20k gas offset for balance updates after fee calculation
1944 | uint256 gasAtStart = gasleft() + s_gasEstimates.Constants.GKEY_OFFSET_RENEW;
1945 |
1946 | SAccounts memory accounts = LibBeacon._resolveAddressCalldata(_addressData);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1970 // Second beacon can renew first if the first beacon has not yet submitted
1971 // Here we check if it's NOT the first allowed renewer, and let anyone else submit after another full expiration period.
1972 !((msg.sender == accounts.beacons[0] && hashes[0] != bytes10(0)) ||
1973 (msg.sender == accounts.beacons[1] && hashes[1] != bytes10(0) && hashes[0] == bytes10(0)))
1974 ) {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1970 // Second beacon can renew first if the first beacon has not yet submitted
1971 // Here we check if it's NOT the first allowed renewer, and let anyone else submit after another full expiration period.
1972 !((msg.sender == accounts.beacons[0] && hashes[0] != bytes10(0)) ||
1973 (msg.sender == accounts.beacons[1] && hashes[1] != bytes10(0) && hashes[0] == bytes10(0)))
1974 ) {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1971 // Here we check if it's NOT the first allowed renewer, and let anyone else submit after another full expiration period.
1972 !((msg.sender == accounts.beacons[0] && hashes[0] != bytes10(0)) ||
1973 (msg.sender == accounts.beacons[1] && hashes[1] != bytes10(0) && hashes[0] == bytes10(0)))
1974 ) {
1975 _expirationHeight += packed.data.expirationBlocks;
```


UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1971 // Here we check if it's NOT the first allowed renewer, and let anyone else submit after another full expiration period.
1972 !((msg.sender == accounts.beacons[0] && hashes[0] != bytes10(0)) ||
1973 (msg.sender == accounts.beacons[1] && hashes[1] != bytes10(0) && hashes[0] == bytes10(0)))
1974 ) {
1975     _expirationHeight += packed.data.expirationBlocks;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1971 // Here we check if it's NOT the first allowed renewer, and let anyone else submit after another full expiration period.
1972 !((msg.sender == accounts.beacons[0] && hashes[0] != bytes10(0)) ||
1973 (msg.sender == accounts.beacons[1] && hashes[1] != bytes10(0) && hashes[0] == bytes10(0)))
1974 ) {
1975     _expirationHeight += packed.data.expirationBlocks;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1991 address[3] memory reqBeacons = accounts.beacons;
1992 for (uint256 i; i < 2; i++) {
1993     if (hashes[i] == bytes10(0) && reqBeacons[i] != address(0)) {
1994         address beaconAddress = reqBeacons[i];
1995         _strikeBeacon(beaconAddress);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1991 | address[3] memory reqBeacons = accounts.beacons;
1992 | for (uint256 i; i < 2; i++) {
1993 |   if (hashes[i] == bytes10(0) && reqBeacons[i] != address(0)) {
1994 |     address beaconAddress = reqBeacons[i];
1995 |     _strikeBeacon(beaconAddress);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1992 | for (uint256 i; i < 2; i++) {
1993 |   if (hashes[i] == bytes10(0) && reqBeacons[i] != address(0)) {
1994 |     address beaconAddress = reqBeacons[i];
1995 |     _strikeBeacon(beaconAddress);
1996 |     beaconsToStrike[i] = beaconAddress;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
1994 | address beaconAddress = reqBeacons[i];
1995 | _strikeBeacon(beaconAddress);
1996 | beaconsToStrike[i] = beaconAddress;
1997 | beaconsToStrikeLen++;
1998 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2002 | // The 3rd beacon is only set if the other 2 have submitted values
2003 | // This beacon never has a stored vrf value (since they're deleted on finalization) so we don't need to check it
2004 | if (reqBeacons[2] != address(0)) {
2005 |     address beaconAddress = reqBeacons[2];
2006 |     _strikeBeacon(beaconAddress);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2003 | // This beacon never has a stored vrf value (since they're deleted on finalization) so we don't need to check it
2004 | if (reqBeacons[2] != address(0)) {
2005 |     address beaconAddress = reqBeacons[2];
2006 |     _strikeBeacon(beaconAddress);
2007 |     beaconsToStrike[2] = beaconAddress;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2005 | address beaconAddress = reqBeacons[2];
2006 | _strikeBeacon(beaconAddress);
2007 | beaconsToStrike[2] = beaconAddress;
2008 | beaconsToStrikeLen++;
2009 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2022 | address firstStrikeBeacon;
2023 | for (uint256 i; i < beaconsToStrike.length; i++) {
2024 |   if (beaconsToStrike[i] == address(0)) continue;
2025 |
2026 |   if (firstStrikeBeacon == address(0)) firstStrikeBeacon = beaconsToStrike[i];
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2024 | if (beaconsToStrike[i] == address(0)) continue;
2025 |
2026 | if (firstStrikeBeacon == address(0)) firstStrikeBeacon = beaconsToStrike[i];
2027 |
2028 | Beacon memory strikeBeacon = s.beacon[beaconsToStrike[i]];
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2026 | if (firstStrikeBeacon == address(0)) firstStrikeBeacon = beaconsToStrike[i];
2027 |
2028 | Beacon memory strikeBeacon = s.beacon[beaconsToStrike[i]];
2029 |
2030 | // If beacon drops below minimum collateral in any token: drop them from beacons list
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2035 | if (  
2036 |   strikeBeacon.registered &&  
2037 |   (s.ethCollateral[beaconsToStrike[i]] < s.configUints[Constants.CKEY_MIN_STAKE_ETH] ||  
2038 |   // tokenCollateral[beaconsToStrike[i]] < minToken ||  
2039 |   strikeBeacon.strikes > s.configUints[Constants.CKEY_MAX_STRIKES])
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2035 | if (  
2036 |   strikeBeacon.registered &&  
2037 |   (s.ethCollateral[beaconsToStrike[i]] < s.configUints[Constants.CKEY_MIN_STAKE_ETH] ||  
2038 |   // tokenCollateral[beaconsToStrike[i]] < minToken ||  
2039 |   strikeBeacon.strikes > s.configUints[Constants.CKEY_MAX_STRIKES])
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2037 | (s.ethCollateral[beaconsToStrike[i]] < s.configUints[Constants.CKEY_MIN_STAKE_ETH] ||  
2038 | // tokenCollateral[beaconsToStrike[i]] < minToken ||  
2039 | strikeBeacon.strikes > s.configUints[Constants.CKEY_MAX_STRIKES])  
2040 | {  
2041 | // Remove beacon from beacons
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2040 | } {
2041 | // Remove beacon from beacons
2042 | _removeBeacon(beaconsToStrike[i]);
2043 | emit Events.UnregisterBeacon(beaconsToStrike[i], true, s.beacon[beaconsToStrike[i]].strikes);
2044 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2041 | // Remove beacon from beacons
2042 | _removeBeacon(beaconsToStrike[i]);
2043 | emit Events.UnregisterBeacon(beaconsToStrike[i], true, s.beacon[beaconsToStrike[i]].strikes);
2044 | }
2045 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2041 | // Remove beacon from beacons
2042 | _removeBeacon(beaconsToStrike[i]);
2043 | emit Events.UnregisterBeacon(beaconsToStrike[i], true, s.beacon[beaconsToStrike[i]].strikes);
2044 | }
2045 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2150 | uint256 i;  
2151 |  
2152 | address[5] memory excludedBeacons = [_beacons[0], _beacons[1], _beacons[2], address(0), address(0)];  
2153 | (address[] memory availableBeacons, uint256 count) = _beaconsWithoutExcluded(_beacons);  
2154 | uint256 excludedBeaconCount = 3;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2150 | uint256 i;  
2151 |  
2152 | address[5] memory excludedBeacons = [_beacons[0], _beacons[1], _beacons[2], address(0), address(0)];  
2153 | (address[] memory availableBeacons, uint256 count) = _beaconsWithoutExcluded(_beacons);  
2154 | uint256 excludedBeaconCount = 3;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2150 | uint256 i;  
2151 |  
2152 | address[5] memory excludedBeacons = [_beacons[0], _beacons[1], _beacons[2], address(0), address(0)];  
2153 | (address[] memory availableBeacons, uint256 count) = _beaconsWithoutExcluded(_beacons);  
2154 | uint256 excludedBeaconCount = 3;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2157 | // If non-submitter
2158 | if (
2159 | (i != 2 && _values[i] == bytes10(0) && _beacons[i] != address(0)) ||
2160 | (i == 2 && _beacons[i] != address(0))
2161 | ) {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2157 | // If non-submitter
2158 | if (
2159 | (i != 2 && _values[i] == bytes10(0) && _beacons[i] != address(0)) ||
2160 | (i == 2 && _beacons[i] != address(0))
2161 | ) {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2158 | if (
2159 | (i != 2 && _values[i] == bytes10(0) && _beacons[i] != address(0)) ||
2160 | (i == 2 && _beacons[i] != address(0))
2161 | ) {
2162 | // Generate new beacon beacon index
```


UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2163 | uint256 randomBeaconIndex = uint256(random) % count;
2164 | // Get a random beacon from the available beacons
2165 | address randomBeacon = availableBeacons[randomBeaconIndex];
2166 | // Assign the random beacon to newSelectedBeacons
2167 | newSelectedBeacons[i] = randomBeacon;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2165 | address randomBeacon = availableBeacons[randomBeaconIndex];
2166 | // Assign the random beacon to newSelectedBeacons
2167 | newSelectedBeacons[i] = randomBeacon;
2168 | s.beacon[randomBeacon].pending++;
2169 | // Add the beacon to the excluded beacons
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2168 | s.beacon[randomBeacon].pending++;
2169 | // Add the beacon to the excluded beacons
2170 | excludedBeacons[excludedBeaconCount] = randomBeacon;
2171 | excludedBeaconCount++;
2172 | // Update the available beacons
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2174 | } else {  
2175 | // If the beacon already submitted, assign it to its existing position  
2176 | newSelectedBeacons[i] = _beacons[i];  
2177 | }  
2178 | unchecked {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2174 | } else {  
2175 | // If the beacon already submitted, assign it to its existing position  
2176 | newSelectedBeacons[i] = _beacons[i];  
2177 | }  
2178 | unchecked {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2273 | function estimateFee(uint256 _callbackGasLimit) public view returns (uint256 estimateFee) {  
2274 | return  
2275 | ((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +  
2276 | _callbackGasLimit +  
2277 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2275 | ((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +  
2276 | _callbackGasLimit +  
2277 | (($gasEstimates.Constants.GKEY_GAS_PER_BEACON_SELECT * (s.beacons.length - 1)) * 3)) *  
2278 | LibNetwork._gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);  
2279 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2276 | _callbackGasLimit +  
2277 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *  
2278 | LibNetwork._gasPrice()) + ($configUints.Constants.CKEY_BEACON_FEE * 5);  
2279 | }  
2280 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2290 | {  
2291 | return  
2292 | (($gasEstimates.Constants.GKEY_TOTAL_SUBMIT +  
2293 | _callbackGasLimit +  
2294 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2292 | ((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +  
2293 | _callbackGasLimit +  
2294 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *  
2295 | _gasPrice) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);  
2296 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2293 | _callbackGasLimit +  
2294 | ((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *  
2295 | _gasPrice) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);  
2296 | }  
2297 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2301 | function request(uint256 _callbackGasLimit) external returns (uint256 id) {  
2302 | // Check if the callback gas limit is within the allowed range  
2303 | uint256 requestMinGasLimit = s.configUints[Constants.CKEY_REQUEST_MIN_GAS_LIMIT];  
2304 | uint256 requestMaxGasLimit = s.configUints[Constants.CKEY_REQUEST_MAX_GAS_LIMIT];  
2305 | if (_callbackGasLimit < requestMinGasLimit || _callbackGasLimit > requestMaxGasLimit)
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2302 | // Check if the callback gas limit is within the allowed range
2303 | uint256 requestMinGasLimit = s.configUints[Constants.CKEY_REQUEST_MIN_GAS_LIMIT];
2304 | uint256 requestMaxGasLimit = s.configUints[Constants.CKEY_REQUEST_MAX_GAS_LIMIT];
2305 | if (_callbackGasLimit < requestMinGasLimit || _callbackGasLimit > requestMaxGasLimit)
2306 | revert CallbackGasLimitOOB(_callbackGasLimit, requestMinGasLimit, requestMaxGasLimit);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2325 | SRandomUintData memory data = SRandomUintData({
2326 |   ethReserved: _estimateFee,
2327 |   beaconFee: s.configUints[Constants.CKEY_BEACON_FEE],
2328 |   height: LibNetwork._blockNumber(),
2329 |   timestamp: block.timestamp,
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2328 | height: LibNetwork._blockNumber(),
2329 | timestamp: block.timestamp,
2330 | expirationBlocks: s.configUints[Constants.CKEY_EXPIRATION_BLOCKS],
2331 | expirationSeconds: s.configUints[Constants.CKEY_EXPIRATION_SECONDS],
2332 | callbackGasLimit: _callbackGasLimit
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2329 | timestamp: block.timestamp,  
2330 | expirationBlocks: s.configUints[Constants.CKEY_EXPIRATION_BLOCKS],  
2331 | expirationSeconds: s.configUints[Constants.CKEY_EXPIRATION_SECONDS],  
2332 | callbackGasLimit: _callbackGasLimit  
2333 | };
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2446 |  
2447 | // Get the minimum required amount of ETH collateral for a beacon  
2448 | uint256 minStakeEth = s.configUints[Constants.CKEY_MIN_STAKE_ETH];  
2449 |  
2450 | // Check if the beacon is already registered
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2483 | // Check if the beacon's collateral is below the minimum required amount  
2484 | if (  
2485 | s.ethCollateral[msg.sender] < s.configUints[Constants.CKEY_MIN_STAKE_ETH] &&  
2486 | s.beaconIndex[msg.sender] != 0  
2487 | ) {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2569 | address(this),
2570 | accounts.client,
2571 | _rsAndSeed[2],
2572 | packed.id,
2573 | packed.vrf.proof,
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2580 | ),
2581 | _v,
2582 | _rsAndSeed[0],
2583 | _rsAndSeed[1]
2584 | );
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2581 | _v,
2582 | _rsAndSeed[0],
2583 | _rsAndSeed[1]
2584 | );
2585 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2585 |  
2586 | // Process the submission for the given beacon  
2587 | _submissionStep(_beacon, beaconPos, _rsAndSeed.2, gasAtStart, packed, accounts);  
2588 | }  
2589 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2634 | ) revert VRFProofInvalid();  
2635 |  
2636 | bytes10 vrfHash = bytes10(keccak256(abi.encodePacked(packed.vrf.proof.0, packed.vrf.proof[1])));  
2637 |  
2638 | // Every 100 consecutive submissions, strikes are reset to 0
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2634 | ) revert VRFProofInvalid();  
2635 |  
2636 | bytes10 vrfHash = bytes10(keccak256(abi.encodePacked(packed.vrf.proof[0], packed.vrf.proof.1)));  
2637 |  
2638 | // Every 100 consecutive submissions, strikes are reset to 0
```


UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2641 |  
2642 | if (beaconPos < 2) {  
2643 |   s.requestToVrfHashes[packed.id][beaconPos] = vrfHash;  
2644 |   reqValues[beaconPos] = vrfHash;  
2645 |   _processRandomSubmission(accounts, packed, gasAtStart, reqValues);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2642 | if (beaconPos < 2) {  
2643 |   s.requestToVrfHashes[packed.id][beaconPos] = vrfHash;  
2644 |   reqValues[beaconPos] = vrfHash;  
2645 |   _processRandomSubmission(accounts, packed, gasAtStart, reqValues);  
2646 | } else {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2664 | packed.id,  
2665 | accounts.client,  
2666 | [reqValues[0], reqValues[1], vrfHash],  
2667 | packed.data.callbackGasLimit,  
2668 | packed.data.ethReserved
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2664 | packed.id,  
2665 | accounts.client,  
2666 | [reqValues[0], reqValues[1], vrfHash],  
2667 | packed.data.callbackGasLimit,  
2668 | packed.data.ethReserved
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2673 | gasAtStart,  
2674 | packed.data.beaconFee,  
2675 | s gasEstimates[Constants.GKEY_OFFSET_FINAL_SUBMIT]  
2676 | );  
2677 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2720 | } private {  
2721 | // Check if the second to last request is valid and non-zero  
2722 | if (reqValues[0] != bytes10(0) && reqValues[1] != bytes10(0)) {  
2723 | bytes10 memBlockhash = bytes10(LibNetwork._blockHash(packed.data.height));  
2724 | if (memBlockhash == bytes10(0)) revert BlockhashUnavailable(packed.data.height);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2720 | } private {
2721 | // Check if the second to last request is valid and non-zero
2722 | if (reqValues[0] != bytes10(0) && reqValues[1] != bytes10(0)) {
2723 | bytes10 memBlockhash = bytes10(LibNetwork._blockHash(packed.data.height));
2724 | if (memBlockhash == bytes10(0)) revert BlockhashUnavailable(packed.data.height);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2724 | if (memBlockhash == bytes10(0)) revert BlockhashUnavailable(packed.data.height);
2725 | // Generate a new seed value using the values of the last two requests + the request's blockhash
2726 | bytes32 newSeed = keccak256(abi.encodePacked(reqValues[0], reqValues[1], memBlockhash));
2727 | // Request the final beacon with the generated seed value
2728 | _requestBeacon(packed.id, 2, newSeed, accounts, packed.data);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2724 | if (memBlockhash == bytes10(0)) revert BlockhashUnavailable(packed.data.height);
2725 | // Generate a new seed value using the values of the last two requests + the request's blockhash
2726 | bytes32 newSeed = keccak256(abi.encodePacked(reqValues[0], reqValues[1], memBlockhash));
2727 | // Request the final beacon with the generated seed value
2728 | _requestBeacon(packed.id, 2, newSeed, accounts, packed.data);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2733 | gasAtStart,  
2734 | packed.data.beaconFee,  
2735 | s.gasEstimates.Constants.GKEY_OFFSET_SUBMIT  
2736 | );  
2737 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2751 | } private view {  
2752 | // Check if the selected beacon is in the correct position in the beacon array  
2753 | if (_beacons.beaconPos != _beacon) revert BeaconNotSelected();  
2754 |  
2755 | // Check if the last two requests are valid (i.e. not the zero value)
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2754 |  
2755 | // Check if the last two requests are valid (i.e. not the zero value)  
2756 | if (beaconPos < 2 && reqValues.beaconPos != bytes10(0)) revert BeaconValueExists();  
2757 |  
2758 | if (msg.sender != _beacon) {
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2818 | /// @notice Returns the value of a contract configuration key
2819 | function configUint(uint256 key) external view returns (uint256) {
2820 |     return s.configUints[key];
2821 | }
2822 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2828 | /// @notice Returns the value of a gas estimate key
2829 | function gasEstimate(uint256 key) external view returns (uint256) {
2830 |     return s.gasEstimates[key];
2831 | }
2832 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2869 | function setConfigUint(uint256 key, uint256 _value) external {
2870 |     LibDiamond.enforceIsContractOwner();
2871 |     emit UpdateContractConfig(key, s.configUints[key], _value);
2872 |     s.configUints[key] = _value;
2873 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2870 | LibDiamond.enforceIsContractOwner();
2871 | emit UpdateContractConfig(key, s.configUints[key], _value);
2872 | s.configUints[key] = _value;
2873 | }
2874 |
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2876 | function setGasEstimate(uint256 key, uint256 _value) external {
2877 |     LibDiamond.enforceIsContractOwner();
2878 |     emit UpdateGasConfig(key, s.gasEstimates[key], _value);
2879 |     s.gasEstimates[key] = _value;
2880 | }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

Entry_flat.sol

Locations

```
2877 | LibDiamond.enforceIsContractOwner();
2878 | emit UpdateGasConfig(key, s.gasEstimates[key], _value);
2879 | s.gasEstimates[key] = _value;
2880 | }
2881 |
```

LOW

Potential use of "blockhash" as source of randomness.

SWC-120

The environment variable "blockhash" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

Entry_flat.sol

Locations

```
12 | address(this),
13 | id,
14 | blockhash.block.number - 1,
15 | block.difficulty,
16 | block.timestamp,
```

LOW

Potential use of "blockhash" as source of randomness.

SWC-120

The environment variable "blockhash" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

Entry_flat.sol

Locations

```
27 |
28 | function _blockHash(uint256 blockNumber) internal view returns (bytes32) {
29 |     return blockhash.blockNumber;
30 | }
31 |
```

LOW

Potential use of "block.number" as source of randomness.

SWC-120

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

Entry_flat.sol

Locations

```
12 | address(this),
13 | id,
14 | blockhash.block.number - 1,
15 | block.difficulty,
16 | block.timestamp,
```

LOW

Potential use of "block.number" as source of randomness.

SWC-120

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

Entry_flat.sol

Locations

```
31 |  
32 | function _blockNumber() internal view returns (uint256) {  
33 |     return block.number;  
34 | }  
35 | }
```