MythX

Analysis d9740916-3d1f-43c3-803c-9f306fd41be0 MythX

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 SWC-101 Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

```
address(this),
id,
blockhash(block.number - 1),
block.difficulty,
block.timestamp,
```

UNKNOWN SWC-101 Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```

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

UNKNOWN

SWC-101

Arithmetic operation "/" discovered

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

Source file

Entry_flat.sol

Locations

```

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

UNKNOWN

SWC-101

Arithmetic operation "/" discovered

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

Source file

Entry_flat.sol

Locations

```
uint256 t;
while (_x != 0) {
t = r / _x;
(q, newT) = (newT, addmod(q, (PP - mulmod(t, newT, PP)), PP));
(r, _x) = (_x, r - t * _x);
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

```
while (_x != 0) {
t = r / _x;
(q, newT) = (newT, addmod(q, (PP - mulmod(t, newT, PP)), PP));
(r, _x) = (_x, r - t * _x);
}
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

```
t = r / _x;
(q, newT) = (newT, addmod(q, (PP - mulmod(t, newT, PP)), PP));
(r, _x) = (_x, r - t * _x);
}

```

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

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

Source file

Entry_flat.sol

Locations

```
t = r / _x;
(q, newT) = (newT, addmod(q, (PP - mulmod(t, newT, PP)), PP));
(r, _x) = (_x, r - t * _x);
}

```

UNKNOWN

SWC-101

Arithmetic operation "/" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "%" discovered

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

Source file

Entry_flat.sol

Locations

```
y2 = expMod(y2, (PP + 1) / 4);
// uint256 cmp = yBit ^ y_ & 1;
uint256 y = (y2 + _prefix) % 2 == 0 ? y2 : PP - y2;

return y;
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```
y2 = expMod(y2, (PP + 1) / 4);
// uint256 cmp = yBit ^ y_ & 1;
uint256 y = (y2 + _prefix) % 2 == 0 ? y2 : PP - y2;

return y;
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

```
y2 = expMod(y2, (PP + 1) / 4);
// uint256 cmp = yBit ^ y_ & 1;
uint256 y = (y2 + _prefix) % 2 == 0 ? y2 : PP - y2;

return y;
```

UNKNOWN

SWC-101

Arithmetic operation "%" discovered

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

Source file

Entry_flat.sol

Locations

```
/// @return (x, -y)
function ecInv(uint256 _x, uint256 _y) internal pure returns (uint256, uint256) {
return (_x, (PP - _y) % PP);
}

```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

```
/// @return (x, -y)
function ecInv(uint256 _x, uint256 _y) internal pure returns (uint256, uint256) {
return (_x, (PP - _y) % PP);
}

```

UNKNOWN Arithmetic operation "-" discovered

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

SWC-101

Source file

Entry_flat.sol

Locations

```
uint256[4] memory hr;
//h
hr[0] = addmod(zs[2], PP - zs[0], PP);
//r
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

```
hr[0] = addmod(zs[2], PP - zs[0], PP);
//r
hr[1] = addmod(zs[3], PP - zs[1], PP);
//h^2
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

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "/" discovered

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

Source file

Entry_flat.sol

Locations

```
(qx, qy, qz) = jacAdd(qx, qy, qz, _x, _y, _z);
}
remaining = remaining / 2;
(_x, _y, _z) = jacDouble(_x, _y, _z);
}
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
}
unchecked {
++ctr;
}
} while (ctr < 256);
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```
/// @return The point coordinates as bytes
function encodePoint(uint256 _x, uint256 _y) internal pure returns (bytes memory) {
uint8 prefix = uint8(2 + (_y % 2));

return abi.encodePacked(prefix, _x);
```

UNKNOWN

SWC-101

Arithmetic operation "%" discovered

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

Source file

`Entry_flat.sol`

Locations

```
/// @return The point coordinates as bytes
function encodePoint(uint256 _x, uint256 _y) internal pure returns (bytes memory) {
uint8 prefix = uint8(2 + (_y % 2));

return abi.encodePacked(prefix, _x);
```

UNKNOWN

SWC-101

Arithmetic operation "%" discovered

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

Source file

`Entry_flat.sol`

Locations

```
uint256 _qy
) internal pure returns (bool) {
address result = ecrecover(0, _y % 2 != 0 ? 28 : 27, bytes32(_x), bytes32(mulmod(_scalar, _x, NN)));

return pointToAddress(_qx, _qy) == result;
```

UNKNOWN

SWC-101

Arithmetic operation "%" discovered

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

Source file

`Entry_flat.sol`

Locations

```
uint256 _qy
) internal pure returns (bool) {
uint256 scalar1 = (NN - _scalar1) % NN;
scalar1 = mulmod(scalar1, _x, NN);
uint256 scalar2 = (NN - _scalar2) % NN;
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

```
uint256 _qy
) internal pure returns (bool) {
uint256 scalar1 = (NN - _scalar1) % NN;
scalar1 = mulmod(scalar1, _x, NN);
uint256 scalar2 = (NN - _scalar2) % NN;
```

UNKNOWN

SWC-101

Arithmetic operation "%" discovered

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

Source file

`Entry_flat.sol`

Locations

```
uint256 scalar1 = (NN - _scalar1) % NN;
scalar1 = mulmod(scalar1, _x, NN);
uint256 scalar2 = (NN - _scalar2) % NN;

address result = ecrecover(
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

```
uint256 scalar1 = (NN - _scalar1) % NN;
scalar1 = mulmod(scalar1, _x, NN);
uint256 scalar2 = (NN - _scalar2) % NN;

address result = ecrecover(
```

UNKNOWN

SWC-101

Arithmetic operation "%" discovered

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

Source file

Entry_flat.sol

Locations

```
address result = ecrecover(
bytes32(scalar1),
_y % 2 != 0 ? 28 : 27,
bytes32(_x),
bytes32(mulmod(scalar2, _x, NN))
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```
uint256 offset
) internal view returns (uint256) {
return _beaconFee + (LibNetwork._gasPrice() * (gasAtStart + offset - gasleft()));
}

```

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

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

Source file

Entry_flat.sol

Locations

```
uint256 offset
) internal view returns (uint256) {
return _beaconFee + (LibNetwork._gasPrice() * (gasAtStart + offset - gasleft()));
}

```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

```
uint256 offset
) internal view returns (uint256) {
return _beaconFee + (LibNetwork._gasPrice() * (gasAtStart + offset - gasleft()));
}

```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```
uint256 offset
) internal view returns (uint256) {
return _beaconFee + (LibNetwork._gasPrice() * (gasAtStart + offset - gasleft()));
}

```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

`Entry_flat.sol`

Locations

```
removeFunction(ds, oldFacetAddress, selector);
addFunction(ds, selector, selectorPosition, _facetAddress);
selectorPosition++;
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "%" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
selectedItems[i] = selectedItems[j];
selectedItems[j] = temp;
s.beacon[selectedItems[i]].pending++;
unchecked {
++i;
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
s.beacon[selectedItems[i]].pending++;
unchecked {
++i;
}
} while (i < 3);
```

UNKNOWN

SWC-101

Arithmetic operation "%" discovered

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

Source file

Entry_flat.sol

Locations

```

// Generate a random index j such that j <= count
uint256 j = uint256(_random) % count;

return selectedItems[j];
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

```
{
uint256 beaconsLen = s.beacons.length;
address[] memory selectedItems = new address[](beaconsLen - 2);

uint256 i = 1;
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
}
unchecked {
++j;
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
selectedItems[count] = s.beacons[i];
unchecked {
++count;
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

`Entry_flat.sol`

Locations

```
}
unchecked {
++i;
}
} while (i < beaconsLen);
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

```
{
uint256 beaconsLen = s.beacons.length;
address[] memory selectedItems = new address[](beaconsLen - 3);

uint256 i = 1;
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

`Entry_flat.sol`

Locations

```
}
unchecked {
++j;
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

`Entry_flat.sol`

Locations

```
selectedItems[count] = s.beacons[i];
unchecked {
++count;
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

`Entry_flat.sol`

Locations

```
}
unchecked {
++i;
}
} while (i < beaconsLen);
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

```
{
uint256 beaconsLen = s.beacons.length;
address[] memory selectedItems = new address[](beaconsLen - excludeLen);

uint256 i = 1;
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
}
unchecked {
++j;
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
selectedItems[count] = s.beacons[i];
unchecked {
++count;
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
}
unchecked {
++i;
}
} while (i < beaconsLen);
```

UNKNOWN SWC-101

Arithmetic operation "-=" discovered

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

Source file

Entry_flat.sol

Locations

```
// Callback to requesting contract
LibBeacon._callback(client, callbackGasLimit, id, result);
s.ethReserved[client] -= _ethReserved;

s.results[id] = result;
```

UNKNOWN SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

`Entry_flat.sol`

Locations

```
fee = deposit;
}
s.requestToFeePaid[id] += fee + seqFee + daoFee;
_chargeClient(client, msg.sender, fee);
}
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```
fee = deposit;
}
s.requestToFeePaid[id] += fee + seqFee + daoFee;
_chargeClient(client, msg.sender, fee);
}
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```
fee = deposit;
}
s.requestToFeePaid[id] += fee + seqFee + daoFee;
_chargeClient(client, msg.sender, fee);
}
```

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

```
fee = deposit;
}
s.requestToFeePaid[id] += fee;
_chargeClient(client, msg.sender, fee);
}
```

UNKNOWN

SWC-101

Arithmetic operation "-=" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```
) external {
// 20k gas offset for balance updates after fee calculation
uint256 gasAtStart = gasleft() + s.gasEstimates[Constants.GKEY_OFFSET_RENEW];

SAccounts memory accounts = LibBeacon._resolveAddressCalldata(_addressData);
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```

{
uint256 _expirationHeight = packed.data.height + packed.data.expirationBlocks;
uint256 _expirationTime = packed.data.timestamp + packed.data.expirationSeconds;
if (msg.sender == s.sequencer) {
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "/" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "/" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

```
) {
_expirationHeight += packed.data.expirationBlocks;
_expirationTime += packed.data.expirationSeconds;
}

```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
_strikeBeacon(beaconAddress);
beaconsToStrike[i] = beaconAddress;
beaconsToStrikeLen++;
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
_strikeBeacon(beaconAddress);
beaconsToStrike[2] = beaconAddress;
beaconsToStrikeLen++;
}

```

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

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

Source file

Entry_flat.sol

Locations

```

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

`Entry_flat.sol`

Locations

```

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

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

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

Source file

`Entry_flat.sol`

Locations

```
revert NotEnoughBeaconsAvailable(
s.beacons.length,
s.beacons.length < 5 ? 5 : beaconsToStrikeLen * 2
);

```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

`Entry_flat.sol`

Locations

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

```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```

// The paying non-submitter might fall below collateral here. It will be removed on next strike if it doesn't add collateral.
uint256 renewFee = packed.data.beaconFee + (LibNetwork._gasPrice() * (gasAtStart - gasleft()));

uint256 refundToClient = s.requestToFeePaid[packed.id];
```

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

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

Source file

Entry_flat.sol

Locations

```

// The paying non-submitter might fall below collateral here. It will be removed on next strike if it doesn't add collateral.
uint256 renewFee = packed.data.beaconFee + (LibNetwork._gasPrice() * (gasAtStart - gasleft()));

uint256 refundToClient = s.requestToFeePaid[packed.id];
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

```

// The paying non-submitter might fall below collateral here. It will be removed on next strike if it doesn't add collateral.
uint256 renewFee = packed.data.beaconFee + (LibNetwork._gasPrice() * (gasAtStart - gasleft()));

uint256 refundToClient = s.requestToFeePaid[packed.id];
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```

uint256 refundToClient = s.requestToFeePaid[packed.id];
uint256 totalCharge = renewFee + refundToClient;

// If charging more than the striked beacon has staked, refund the remaining stake to the client
```

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

```
totalCharge = firstCollateral;
renewFee = renewFee > totalCharge ? totalCharge : renewFee;
s.ethCollateral[msg.sender] += renewFee;
emit Events.ChargeEth(
firstStrikeBeacon,
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

```
refundToClient = totalCharge - renewFee;
if (refundToClient > 0) {
s.ethDeposit[accounts.client] += refundToClient;
emit Events.ChargeEth(
firstStrikeBeacon,
```

UNKNOWN

SWC-101

Arithmetic operation "-=" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

```
s.ethCollateral[firstStrikeBeacon] -= totalCharge;
// Refund this function's gas to the caller
s.ethCollateral[msg.sender] += renewFee;
s.ethDeposit[accounts.client] += refundToClient;
// Add to fees refunded
```

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
function _strikeBeacon(address _beacon) internal {
Beacon memory tempBeacon = s.beacon[_beacon];
if (tempBeacon.registered) tempBeacon.strikes++;
tempBeacon.consecutiveSubmissions = 0;
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

```
if (tempBeacon.registered) tempBeacon.strikes++;
tempBeacon.consecutiveSubmissions = 0;
if (tempBeacon.pending > 0) tempBeacon.pending--;
s.beacon[_beacon] = tempBeacon;
}
```

UNKNOWN Arithmetic operation "-" discovered

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

SWC-101

Source file

Entry_flat.sol

Locations

```
address(this),
_request,
LibNetwork._blockHash(LibNetwork._blockNumber() - 1),
block.chainid
)
```

UNKNOWN Arithmetic operation "%" discovered

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

SWC-101

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
// Assign the random beacon to newSelectedBeacons
newSelectedBeacons[i] = randomBeacon;
s.beacon[randomBeacon].pending++;
// Add the beacon to the excluded beacons
excludedBeacons[excludedBeaconCount] = randomBeacon;
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```
}
unchecked {
++i;
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

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

```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

```
// Check if the client is trying to withdraw more than they have deposited
if (_amount > s.ethDeposit[msg.sender] - s.ethReserved[msg.sender])
revert WithdrawingTooMuch(_amount, s.ethDeposit[msg.sender] - s.ethReserved[msg.sender]);

// Decrease the client's deposit by the amount they are withdrawing
```

UNKNOWN

SWC-101

Arithmetic operation "-=" discovered

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

Source file

`Entry_flat.sol`

Locations

```

// Decrease the client's deposit by the amount they are withdrawing
s.ethDeposit[msg.sender] -= _amount;

// Emit an event to log the withdrawal
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

`Entry_flat.sol`

Locations

```
function estimateFee(uint256 _callbackGasLimit) public view returns (uint256 esimateFee) {
return
((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
_callbackGasLimit +
((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
LibNetwork._gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
}

```

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

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

Source file

`Entry_flat.sol`

Locations

```
function estimateFee(uint256 _callbackGasLimit) public view returns (uint256 esimateFee) {
return
((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
_callbackGasLimit +
((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
LibNetwork._gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
}

```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```
function estimateFee(uint256 _callbackGasLimit) public view returns (uint256 esimateFee) {
return
((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
_callbackGasLimit +
((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
LibNetwork._gasPrice()) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
}
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

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

Source file

Entry_flat.sol

Locations

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

```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

`Entry_flat.sol`

Locations

```
{
return
((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
_callbackGasLimit +
((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
_gasPrice) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
}

```

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

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

Source file

`Entry_flat.sol`

Locations

```
{
return
((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
_callbackGasLimit +
((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
_gasPrice) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
}

```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

`Entry_flat.sol`

Locations

```
{
return
((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
_callbackGasLimit +
((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
_gasPrice) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
}
```

UNKNOWN SWC-101

Arithmetic operation "+" discovered

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

Source file

Entry_flat.sol

Locations

```
{
return
((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
_callbackGasLimit +
((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
_gasPrice) + (s.configUints[Constants.CKEY_BEACON_FEE] * 5);
```

UNKNOWN SWC-101

Arithmetic operation "*" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN SWC-101

Arithmetic operation "*" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Arithmetic operation "*" discovered

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

Source file

Entry_flat.sol

Locations

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

```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

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

Source file

Entry_flat.sol

Locations

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

```

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

```

// Increase the client's reserved ETH by the estimated fee
s.ethReserved[msg.sender] += _estimateFee;

// Increment the latest request ID and store it in the `id` variable
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

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

Source file

Entry_flat.sol

Locations

```

// Increment the latest request ID and store it in the `id` variable
s.latestRequestId++;
id = s.latestRequestId;

```

UNKNOWN

SWC-101

Arithmetic operation "+=" discovered

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

Source file

Entry_flat.sol

Locations

```
function beaconStakeEth(address _beacon) external payable {
// Increase the beacon's ETH collateral by the value of the transaction
s.ethCollateral[_beacon] += msg.value;

// 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

```
function beaconUnstakeEth(uint256 _amount) external {
// Decrease the beacon's ETH collateral by the specified amount
s.ethCollateral[msg.sender] -= _amount;

// 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

```
// If the consecutive submissions count is less than the maximum allowed, increment it
unchecked {
memBeacon.consecutiveSubmissions++;
}
}
```

UNKNOWN Arithmetic operation "--" discovered

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

SWC-101

Source file

Entry_flat.sol

Locations

```

// Decrement the pending count for the beacon
if (memBeacon.pending > 0) memBeacon.pending--;

// Save the updated Beacon struct
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

`Entry_flat.sol`

Locations

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

// Calculate the earliest block number that the sequencer can submit on behalf of the beacon
```

UNKNOWN

SWC-101

Arithmetic operation "/" discovered

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

Source file

`Entry_flat.sol`

Locations

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

// Calculate the earliest block number that the sequencer can submit on behalf of the beacon
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

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

Source file

`Entry_flat.sol`

Locations

```

// Calculate the earliest block number that the sequencer can submit on behalf of the beacon
uint256 sequencerSubmitBlock = data.height + (data.expirationBlocks / 2);

// Check if the sequencer is submitting too early
```

UNKNOWN

SWC-101

Arithmetic operation "/" discovered

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

Source file

Entry_flat.sol

Locations

```

// Calculate the earliest block number that the sequencer can submit on behalf of the beacon
uint256 sequencerSubmitBlock = data.height + (data.expirationBlocks / 2);

// Check if the sequencer is submitting too early
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

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

Source file

Entry_flat.sol

Locations

```
address(this),
id,
blockhash(block.number - 1),
block.difficulty,
block.timestamp,
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

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

Source file

`Entry_flat.sol`

Locations

```

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

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

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

Source file

`Entry_flat.sol`

Locations

```
address(this),
_request,
LibNetwork._blockHash(LibNetwork._blockNumber() - 1),
block.chainid
)
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

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

Source file

`Entry_flat.sol`

Locations

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

LOW

SWC-103

A floating pragma is set.

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

```

// File: contracts/libraries/LibNetwork.sol
pragma solidity ^0.8.17;


```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
// z1^2, z1^3, z2^2, z2^3
zs[0] = mulmod(_z1, _z1, PP);
zs[1] = mulmod(_z1, zs[0], PP);
zs[2] = mulmod(_z2, _z2, PP);
zs[3] = mulmod(_z2, zs[2], PP);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
// z1^2, z1^3, z2^2, z2^3
zs[0] = mulmod(_z1, _z1, PP);
zs[1] = mulmod(_z1, zs[0], PP);
zs[2] = mulmod(_z2, _z2, PP);
zs[3] = mulmod(_z2, zs[2], PP);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
zs[0] = mulmod(_z1, _z1, PP);
zs[1] = mulmod(_z1, zs[0], PP);
zs[2] = mulmod(_z2, _z2, PP);
zs[3] = mulmod(_z2, zs[2], PP);

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
zs[1] = mulmod(_z1, zs[0], PP);
zs[2] = mulmod(_z2, _z2, PP);
zs[3] = mulmod(_z2, zs[2], PP);

// u1, s1, u2, s2
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
zs[1] = mulmod(_z1, zs[0], PP);
zs[2] = mulmod(_z2, _z2, PP);
zs[3] = mulmod(_z2, zs[2], PP);

// u1, s1, u2, s2
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```

// u1, s1, u2, s2
zs = [mulmod(_x1, zs[2], PP), mulmod(_y1, zs[3], PP), mulmod(_x2, zs[0], PP), mulmod(_y2, zs[1], PP)];

// In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```

// u1, s1, u2, s2
zs = [mulmod(_x1, zs[2], PP), mulmod(_y1, zs[3], PP), mulmod(_x2, zs[0], PP), mulmod(_y2, zs[1], PP)];

// In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
// u1, s1, u2, s2
zs = [mulmod(_x1, zs[2], PP), mulmod(_y1, zs[3], PP), mulmod(_x2, zs[0], PP), mulmod(_y2, zs[1], PP)];

// In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
// u1, s1, u2, s2
zs = [mulmod(_x1, zs[2], PP), mulmod(_y1, zs[3], PP), mulmod(_x2, zs[0], PP), mulmod(_y2, zs[1], PP)];

// In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```

// In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
require(zs[0] != zs[2] || zs[1] != zs[3], "Use jacDouble function instead");

uint256[4] memory hr;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```

// In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
require(zs[0] != zs[2] || zs[1] != zs[3], "Use jacDouble function instead");

uint256[4] memory hr;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```

// In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
require(zs[0] != zs[2] || zs[1] != zs[3], "Use jacDouble function instead");

uint256[4] memory hr;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```

// In case of zs[0] == zs[2] && zs[1] == zs[3], double function should be used
require(zs[0] != zs[2] || zs[1] != zs[3], "Use jacDouble function instead");

uint256[4] memory hr;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256[4] memory hr;
//h
hr[0] = addmod(zs[2], PP - zs[0], PP);
//r
hr[1] = addmod(zs[3], PP - zs[1], PP);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256[4] memory hr;
//h
hr[0] = addmod(zs[2], PP - zs[0], PP);
//r
hr[1] = addmod(zs[3], PP - zs[1], PP);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256[4] memory hr;
//h
hr[0] = addmod(zs[2], PP - zs[0], PP);
//r
hr[1] = addmod(zs[3], PP - zs[1], PP);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
hr[0] = addmod(zs[2], PP - zs[0], PP);
//r
hr[1] = addmod(zs[3], PP - zs[1], PP);
//h^2
hr[2] = mulmod(hr[0], hr[0], PP);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
hr[0] = addmod(zs[2], PP - zs[0], PP);
//r
hr[1] = addmod(zs[3], PP - zs[1], PP);
//h^2
hr[2] = mulmod(hr[0], hr[0], PP);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
hr[0] = addmod(zs[2], PP - zs[0], PP);
//r
hr[1] = addmod(zs[3], PP - zs[1], PP);
//h^2
hr[2] = mulmod(hr[0], hr[0], PP);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
hr[1] = addmod(zs[3], PP - zs[1], PP);
//h^2
hr[2] = mulmod(hr[0], hr[0], PP);
// h^3
hr[3] = mulmod(hr[2], hr[0], PP);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
hr[1] = addmod(zs[3], PP - zs[1], PP);
//h^2
hr[2] = mulmod(hr[0], hr[0], PP);
// h^3
hr[3] = mulmod(hr[2], hr[0], PP);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
hr[1] = addmod(zs[3], PP - zs[1], PP);
//h^2
hr[2] = mulmod(hr[0], hr[0], PP);
// h^3
hr[3] = mulmod(hr[2], hr[0], PP);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
GX,
GY,
_proof[2],
_publicKey[0],
_publicKey[1]
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
GY,
_proof[2],
_publicKey[0],
_publicKey[1]
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
_proof[2],
_publicKey[0],
_publicKey[1]
);

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
hPointX,
hPointY,
_proof[2],
_proof[0],
_proof[1]
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
hPointY,
_proof[2],
_proof[0],
_proof[1]
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
_proof[2],
_proof[0],
_proof[1]
);

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
hPointX,
hPointY,
_proof[0],
_proof[1],
uPointX,
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
hPointY,
_proof[0],
_proof[1],
uPointX,
uPointY,
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```

// Step 6: Check validity c == c'
return uint128(derivedC) == _proof[2];
}

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
if (
!ecMulSubMulVerify(
_proof[3], //s
_proof[2], //c
_publicKey[0], //Y-x
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
!ecMulSubMulVerify(
_proof[3], //s
_proof[2], //c
_publicKey[0], //Y-x
_publicKey[1], //Y-y
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
_proof[3], //s
_proof[2], //c
_publicKey[0], //Y-x
_publicKey[1], //Y-y
_uPoint[0], //U-x
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
_proof[2], //c
_publicKey[0], //Y-x
_publicKey[1], //Y-y
_uPoint[0], //U-x
_uPoint[1]
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
_publicKey[0], //Y-x
_publicKey[1], //Y-y
_uPoint[0], //U-x
_uPoint[1]
) || //U-y
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
_publicKey[1], //Y-y
_uPoint[0], //U-x
_uPoint[1]
) || //U-y
!ecMulVerify(
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
hPointX, //H-x
hPointY, //H-y
_vComponents[0], //s*H -x
_vComponents[1]
) || //s*H -y
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
!ecMulVerify(
_proof[2], //c
_proof[0], //gamma-x
_proof[1], //gamma-y
_vComponents[2], //c*Gamma -x
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
_proof[2], //c
_proof[0], //gamma-x
_proof[1], //gamma-y
_vComponents[2], //c*Gamma -x
_vComponents[3]
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
_proof[0], //gamma-x
_proof[1], //gamma-y
_vComponents[2], //c*Gamma -x
_vComponents[3]
) //c*Gamma -y
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
_proof[1], //gamma-y
_vComponents[2], //c*Gamma -x
_vComponents[3]
) //c*Gamma -y
) {
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
_vComponents[0], //s*H -x
_vComponents[1], //s*H -y
_vComponents[2], //c*Gamma -x
_vComponents[3] //c*Gamma -y
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
_vComponents[1], //s*H -y
_vComponents[2], //c*Gamma -x
_vComponents[3] //c*Gamma -y
);

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
hPointX,
hPointY,
_proof[0],
_proof[1],
_uPoint[0],
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
hPointY,
_proof[0],
_proof[1],
_uPoint[0],
_uPoint[1],
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
_proof[0],
_proof[1],
_uPoint[0],
_uPoint[1],
vPointX,
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
_proof[1],
_uPoint[0],
_uPoint[1],
vPointX,
vPointY
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```

// Step 6: Check validity c == c'
return uint128(derivedC) == _proof[2];
}

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
(uint256 hPointX, uint256 hPointY) = hashToTryAndIncrement(_publicKey, _message);
(uint256 uPointX, uint256 uPointY) = ecMulSubMul(
_proof[3],
GX,
GY,
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
GX,
GY,
_proof[2],
_publicKey[0],
_publicKey[1]
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
GY,
_proof[2],
_publicKey[0],
_publicKey[1]
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

return ([uPointX, uPointY], [sHX, sHY, cGammaX, cGammaY]);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

return ([uPointX, uPointY], [sHX, sHY, cGammaX, cGammaY]);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

return ([uPointX, uPointY], [sHX, sHY, cGammaX, cGammaY]);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint8(1),
// Public Key
encodePoint(_publicKey[0], _publicKey[1]),
// Message
_message
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint8(1),
// Public Key
encodePoint(_publicKey[0], _publicKey[1]),
// Message
_message
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
return
SPackedSubmitData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),
SFastVerifyData(
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
SPackedSubmitData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),
SFastVerifyData(
[_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
SPackedSubmitData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),
SFastVerifyData(
[_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
SPackedSubmitData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),
SFastVerifyData(
[_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
SPackedSubmitData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),
SFastVerifyData(
[_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
SPackedSubmitData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),
SFastVerifyData(
[_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
SPackedSubmitData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),
SFastVerifyData(
[_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
SPackedSubmitData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7]),
SFastVerifyData(
[_data[8], _data[9], _data[10], _data[11]],
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
SFastVerifyData(
[_data[8], _data[9], _data[10], _data[11]],
[_data[12], _data[13]],
[_data[14], _data[15], _data[16], _data[17]]
)
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
SFastVerifyData(
[_data[8], _data[9], _data[10], _data[11]],
[_data[12], _data[13]],
[_data[14], _data[15], _data[16], _data[17]]
)
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
[_data[8], _data[9], _data[10], _data[11]],
[_data[12], _data[13]],
[_data[14], _data[15], _data[16], _data[17]]
)
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
[_data[8], _data[9], _data[10], _data[11]],
[_data[12], _data[13]],
[_data[14], _data[15], _data[16], _data[17]]
)
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
[_data[8], _data[9], _data[10], _data[11]],
[_data[12], _data[13]],
[_data[14], _data[15], _data[16], _data[17]]
)
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
[_data[8], _data[9], _data[10], _data[11]],
[_data[12], _data[13]],
[_data[14], _data[15], _data[16], _data[17]]
)
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
return
SPackedUintData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
SPackedUintData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])
);
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
SPackedUintData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])
);
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
SPackedUintData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])
);
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
SPackedUintData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])
);
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
SPackedUintData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])
);
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
SPackedUintData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])
);
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
SPackedUintData(
uint256(_data[0]),
SRandomUintData(_data[1], _data[2], _data[3], _data[4], _data[5], _data[6], _data[7])
);
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
) internal {
for (uint256 facetIndex; facetIndex < _diamondCut.length; facetIndex++) {
IDiamondCut.FacetCutAction action = _diamondCut[facetIndex].action;
if (action == IDiamondCut.FacetCutAction.Add) {
addFunctions(_diamondCut[facetIndex].facetAddress, _diamondCut[facetIndex].functionSelectors);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
} else if (action == IDiamondCut.FacetCutAction.Replace) {
replaceFunctions(
_diamondCut[facetIndex].facetAddress,
_diamondCut[facetIndex].functionSelectors
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
replaceFunctions(
_diamondCut[facetIndex].facetAddress,
_diamondCut[facetIndex].functionSelectors
);
} else if (action == IDiamondCut.FacetCutAction.Remove) {
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
} else if (action == IDiamondCut.FacetCutAction.Remove) {
removeFunctions(
_diamondCut[facetIndex].facetAddress,
_diamondCut[facetIndex].functionSelectors
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
removeFunctions(
_diamondCut[facetIndex].facetAddress,
_diamondCut[facetIndex].functionSelectors
);
} else {
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
// if not the same then replace _selector with lastSelector
if (selectorPosition != lastSelectorPosition) {
bytes4 lastSelector = ds.facetFunctionSelectors[_facetAddress].functionSelectors[
lastSelectorPosition
];
ds.facetFunctionSelectors[_facetAddress].functionSelectors[selectorPosition] = lastSelector;
ds.selectorToFacetAndPosition[lastSelector].functionSelectorPosition = uint96(selectorPosition);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
lastSelectorPosition
];
ds.facetFunctionSelectors[_facetAddress].functionSelectors[selectorPosition] = lastSelector;
ds.selectorToFacetAndPosition[lastSelector].functionSelectorPosition = uint96(selectorPosition);
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256 facetAddressPosition = ds.facetFunctionSelectors[_facetAddress].facetAddressPosition;
if (facetAddressPosition != lastFacetAddressPosition) {
address lastFacetAddress = ds.facetAddresses[lastFacetAddressPosition];
ds.facetAddresses[facetAddressPosition] = lastFacetAddress;
ds.facetFunctionSelectors[lastFacetAddress].facetAddressPosition = facetAddressPosition;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
if (facetAddressPosition != lastFacetAddressPosition) {
address lastFacetAddress = ds.facetAddresses[lastFacetAddressPosition];
ds.facetAddresses[facetAddressPosition] = lastFacetAddress;
ds.facetFunctionSelectors[lastFacetAddress].facetAddressPosition = facetAddressPosition;
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
return;
}
s.beacons[index] = s.beacons[lastBeaconIndex];
address newBeacon = s.beacons[lastBeaconIndex];
s.beaconIndex[_beacon] = 0;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
return;
}
s.beacons[index] = s.beacons[lastBeaconIndex];
address newBeacon = s.beacons[lastBeaconIndex];
s.beaconIndex[_beacon] = 0;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
// Swap the items at indices i and j
address temp = selectedItems[i];
selectedItems[i] = selectedItems[j];
selectedItems[j] = temp;
s.beacon[selectedItems[i]].pending++;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
// Swap the items at indices i and j
address temp = selectedItems[i];
selectedItems[i] = selectedItems[j];
selectedItems[j] = temp;
s.beacon[selectedItems[i]].pending++;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
address temp = selectedItems[i];
selectedItems[i] = selectedItems[j];
selectedItems[j] = temp;
s.beacon[selectedItems[i]].pending++;
unchecked {
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
selectedItems[i] = selectedItems[j];
selectedItems[j] = temp;
s.beacon[selectedItems[i]].pending++;
unchecked {
++i;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256 j = uint256(_random) % count;

return selectedItems[j];
}

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256 j = 0;
while (j < _excluded.length) {
if (s.beacons[i] == _excluded[j]) {
found = true;
break;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
uint256 j = 0;
while (j < _excluded.length) {
if (s.beacons[i] == _excluded[j]) {
found = true;
break;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
}
if (!found) {
selectedItems[count] = s.beacons[i];
unchecked {
++count;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
}
if (!found) {
selectedItems[count] = s.beacons[i];
unchecked {
++count;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256 j = 0;
while (j < _excluded.length) {
if (s.beacons[i] == _excluded[j]) {
found = true;
break;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256 j = 0;
while (j < _excluded.length) {
if (s.beacons[i] == _excluded[j]) {
found = true;
break;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
}
if (!found) {
selectedItems[count] = s.beacons[i];
unchecked {
++count;
```

UNKNOWN SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
}
if (!found) {
selectedItems[count] = s.beacons[i];
unchecked {
++count;
```

UNKNOWN SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256 j = 0;
while (j < _excluded.length) {
if (s.beacons[i] == _excluded[j]) {
found = true;
break;
```

UNKNOWN SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256 j = 0;
while (j < _excluded.length) {
if (s.beacons[i] == _excluded[j]) {
found = true;
break;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
}
if (!found) {
selectedItems[count] = s.beacons[i];
unchecked {
++count;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
}
if (!found) {
selectedItems[count] = s.beacons[i];
unchecked {
++count;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256 _ethReserved
) internal {
bytes32 result = keccak256(abi.encodePacked(hashes[0], hashes[1], hashes[2]));

// Callback to requesting contract
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256 _ethReserved
) internal {
bytes32 result = keccak256(abi.encodePacked(hashes[0], hashes[1], hashes[2]));

// Callback to requesting contract
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
uint256 _ethReserved
) internal {
bytes32 result = keccak256(abi.encodePacked(hashes[0], hashes[1], hashes[2]));

// Callback to requesting contract
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
) external {
// 20k gas offset for balance updates after fee calculation
uint256 gasAtStart = gasleft() + s.gasEstimates[Constants.GKEY_OFFSET_RENEW];

SAccounts memory accounts = LibBeacon._resolveAddressCalldata(_addressData);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN Out of bounds array access

SWC-110

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN Out of bounds array access

SWC-110

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN Out of bounds array access

SWC-110

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

Source file

Entry_flat.sol

Locations

```
address beaconAddress = reqBeacons[i];
_strikeBeacon(beaconAddress);
beaconsToStrike[i] = beaconAddress;
beaconsToStrikeLen++;
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
address beaconAddress = reqBeacons[2];
_strikeBeacon(beaconAddress);
beaconsToStrike[2] = beaconAddress;
beaconsToStrikeLen++;
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
address firstStrikeBeacon;
for (uint256 i; i < beaconsToStrike.length; i++) {
if (beaconsToStrike[i] == address(0)) continue;

if (firstStrikeBeacon == address(0)) firstStrikeBeacon = beaconsToStrike[i];
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
if (beaconsToStrike[i] == address(0)) continue;

if (firstStrikeBeacon == address(0)) firstStrikeBeacon = beaconsToStrike[i];

Beacon memory strikeBeacon = s.beacon[beaconsToStrike[i]];
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
if (firstStrikeBeacon == address(0)) firstStrikeBeacon = beaconsToStrike[i];

Beacon memory strikeBeacon = s.beacon[beaconsToStrike[i]];

// If beacon drops below minimum collateral in any token: drop them from beacons list
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
// Remove beacon from beacons
_removeBeacon(beaconsToStrike[i]);
emit Events.UnregisterBeacon(beaconsToStrike[i], true, s.beacon[beaconsToStrike[i]].strikes);
}
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
// Remove beacon from beacons
_removeBeacon(beaconsToStrike[i]);
emit Events.UnregisterBeacon(beaconsToStrike[i], true, s.beacon[beaconsToStrike[i]].strikes);
}
}
```

UNKNOWN
SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
uint256 i;

address[5] memory excludedBeacons = [_beacons[0], _beacons[1], _beacons[2], address(0), address(0)];
(address[] memory availableBeacons, uint256 count) = _beaconsWithoutExcluded(_beacons);
uint256 excludedBeaconCount = 3;
```

UNKNOWN
SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
uint256 i;

address[5] memory excludedBeacons = [_beacons[0], _beacons[1], _beacons[2], address(0), address(0)];
(address[] memory availableBeacons, uint256 count) = _beaconsWithoutExcluded(_beacons);
uint256 excludedBeaconCount = 3;
```

UNKNOWN
SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
uint256 i;

address[5] memory excludedBeacons = [_beacons[0], _beacons[1], _beacons[2], address(0), address(0)];
(address[] memory availableBeacons, uint256 count) = _beaconsWithoutExcluded(_beacons);
uint256 excludedBeaconCount = 3;
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
s.beacon[randomBeacon].pending++;
// Add the beacon to the excluded beacons
excludedBeacons[excludedBeaconCount] = randomBeacon;
excludedBeaconCount++;
// Update the available beacons
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
function estimateFee(uint256 _callbackGasLimit) public view returns (uint256 esimateFee) {
return
((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
_callbackGasLimit +
((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
{
return
((s.gasEstimates[Constants.GKEY_TOTAL_SUBMIT] +
_callbackGasLimit +
((s.gasEstimates[Constants.GKEY_GAS_PER_BEACON_SELECT] * (s.beacons.length - 1)) * 3)) *
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
height: LibNetwork._blockNumber(),
timestamp: block.timestamp,
expirationBlocks: s.configUints[Constants.CKEY_EXPIRATION_BLOCKS],
expirationSeconds: s.configUints[Constants.CKEY_EXPIRATION_SECONDS],
callbackGasLimit: _callbackGasLimit
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
timestamp: block.timestamp,
expirationBlocks: s.configUints[Constants.CKEY_EXPIRATION_BLOCKS],
expirationSeconds: s.configUints[Constants.CKEY_EXPIRATION_SECONDS],
callbackGasLimit: _callbackGasLimit
});
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```

// Get the minimum required amount of ETH collateral for a beacon
uint256 minStakeEth = s.configUints[Constants.CKEY_MIN_STAKE_ETH];

// Check if the beacon is already registered
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
address(this),
accounts.client,
_rsAndSeed[2],
packed.id,
packed.vrf.proof,
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
),
_v,
_rsAndSeed[0],
_rsAndSeed[1]
);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
_v,
_rsAndSeed[0],
_rsAndSeed[1]
);

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```

// Process the submission for the given beacon
_submissionStep(_beacon, beaconPos, _rsAndSeed[2], gasAtStart, packed, accounts);
}

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
) revert VRFProofInvalid();

bytes10 vrfHash = bytes10(keccak256(abi.encodePacked(packed.vrf.proof[0], packed.vrf.proof[1])));

// Every 100 consecutive submissions, strikes are reset to 0
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
) revert VRFProofInvalid();

bytes10 vrfHash = bytes10(keccak256(abi.encodePacked(packed.vrf.proof[0], packed.vrf.proof[1])));

// Every 100 consecutive submissions, strikes are reset to 0
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```

if (beaconPos < 2) {
s.requestToVrfHashes[packed.id][beaconPos] = vrfHash;
reqValues[beaconPos] = vrfHash;
_processRandomSubmission(accounts, packed, gasAtStart, reqValues);
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
packed.id,
accounts.client,
[reqValues[0], reqValues[1], vrfHash],
packed.data.callbackGasLimit,
packed.data.ethReserved
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
packed.id,
accounts.client,
[reqValues[0], reqValues[1], vrfHash],
packed.data.callbackGasLimit,
packed.data.ethReserved
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
gasAtStart,
packed.data.beaconFee,
s.gasEstimates[Constants.GKEY_OFFSET_FINAL_SUBMIT]
);

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

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

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
gasAtStart,
packed.data.beaconFee,
s.gasEstimates[Constants.GKEY_OFFSET_SUBMIT]
);

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
) private view {
// Check if the selected beacon is in the correct position in the beacon array
if (_beacons[beaconPos] != _beacon) revert BeaconNotSelected();

// Check if the last two requests are valid (i.e. not the zero value)
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```

// Check if the last two requests are valid (i.e. not the zero value)
if (beaconPos < 2 && reqValues[beaconPos] != bytes10(0)) revert BeaconValueExists();

if (msg.sender != _beacon) {
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

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

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

`Entry_flat.sol`

Locations

```
function setConfigUint(uint256 key, uint256 _value) external {
LibDiamond.enforceIsContractOwner();
emit UpdateContractConfig(key, s.configUints[key], _value);
s.configUints[key] = _value;
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
LibDiamond.enforceIsContractOwner();
emit UpdateContractConfig(key, s.configUints[key], _value);
s.configUints[key] = _value;
}

```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
function setGasEstimate(uint256 key, uint256 _value) external {
LibDiamond.enforceIsContractOwner();
emit UpdateGasConfig(key, s.gasEstimates[key], _value);
s.gasEstimates[key] = _value;
}
```

UNKNOWN

SWC-110

Out of bounds array access

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

Source file

Entry_flat.sol

Locations

```
LibDiamond.enforceIsContractOwner();
emit UpdateGasConfig(key, s.gasEstimates[key], _value);
s.gasEstimates[key] = _value;
}

```

LOW

SWC-120

Potential use of "blockhash" as source of randonmness.

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

```
address(this),
id,
blockhash(block.number - 1),
block.difficulty,
block.timestamp,
```

LOW

SWC-120

Potential use of "blockhash" as source of randonmness.

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

```

function _blockHash(uint256 blockNumber) internal view returns (bytes32) {
return blockhash(blockNumber);
}

```

LOW

SWC-120

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

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

```
address(this),
id,
blockhash(block.number - 1),
block.difficulty,
block.timestamp,
```

LOW

SWC-120

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

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

```

function _blockNumber() internal view returns (uint256) {
return block.number;
}
}
```