



REPORT 5F03C264097E2D0019C30C0E

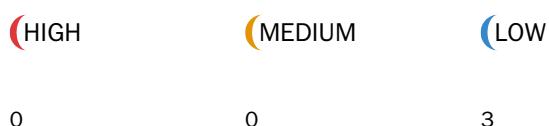
Created Tue Jul 07 2020 00:31:32 GMT+0000 (Coordinated Universal Time)
Number of analyses 6
User goncalo05@gmail.com

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
6df956e5-3745-4600-ba16-734b214c8d16	src/Loihi.sol	3
e3933c9e-1e8c-4fa8-bac4-3886c1b5d906	src/LoihiDelegators.sol	1
6147bc39-5f33-4e3a-8b2e-189d40b5e990	src/LoihiERC20.sol	9
a26303f1-eb49-420f-ae51-5728493c1e06	src/LoihiExchange.sol	6
94aef017-b727-4d0e-b9b8-33e3477b9e1f	src/LoihiLiquidity.sol	12
7f8a2fa5-d6d7-4388-bc74-bf273a7a3f9e	src/LoihiViews.sol	27

Started	Tue Jul 07 2020 00:31:52 GMT+0000 (Coordinated Universal Time)
Finished	Tue Jul 07 2020 01:17:05 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.19
Main Source File	Src/Loihi.sol

DETECTED VULNERABILITIES



ISSUES

LOW A floating pragma is set.

The current pragma Solidity directive is `""^0.5.15""`. 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.

SWC-103

Source file
src/Loihi.sol
Locations

```
12 // along with this program. If not, see <http://www.gnu.org/licenses/>.  
13  
14 pragma solidity ^0.5.15;  
15  
16 import "./LoihiRoot.sol";
```

LOW Function parameter shadows a state variable.

The function parameter "owner" in contract "Loihi" shadows the state variable with the same name "owner" in contract "LoihiRoot".

SWC-119

Source file
src/Loihi.sol
Locations

```
327 }  
328  
329 function allowance (address owner, address spender) public view returns (uint256) {  
330     return allowances[owner][spender];  
331 }
```

LOW

Unused local variable "returndata".

The local variable "returndata" is declared within the function "safeApprove" of contract "Loihi" but its value does not seem to be used anywhere in "safeApprove".

SWC-131

Source file

src/Loihi.sol

Locations

```
352
353 | function safeApprove(ERC20Approve token, address spender, uint256 value) private {
354 |     (bool success, bytes memory returndata) = address(token).call(abi.encodeWithSelector(token.approve.selector, spender, value));
355 |     require(success, "SafeERC20: low-level call failed");
356 }
```

Started	Tue Jul 07 2020 00:31:52 GMT+0000 (Coordinated Universal Time)
Finished	Tue Jul 07 2020 01:17:02 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.19
Main Source File	Src/LoihiDelegators.sol

DETECTED VULNERABILITIES



ISSUES

LOW

A floating pragma is set.

The current pragma Solidity directive is `""^0.5.15""`. 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.

SWC-103

Source file

src/LoihiDelegators.sol

Locations

```
12 // along with this program. If not, see <http://www.gnu.org/licenses/>.  
13  
14 pragma solidity ^0.5.15;  
15  
16 contract LoihiDelegators {
```

Started Tue Jul 07 2020 00:31:52 GMT+0000 (Coordinated Universal Time)

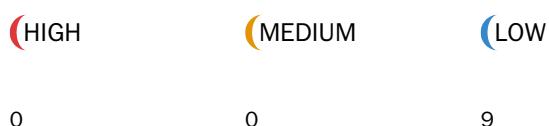
Finished Tue Jul 07 2020 01:17:04 GMT+0000 (Coordinated Universal Time)

Mode Deep

Client Tool Mythx-Cli-0.6.19

Main Source File Src/LoihiERC20.sol

DETECTED VULNERABILITIES



ISSUES

LOW A floating pragma is set.

The current pragma Solidity directive is `""^0.5.15""`. 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.

SWC-103 Source file

src/LoihiERC20.sol

Locations

```
12 // along with this program. If not, see <http://www.gnu.org/licenses/>.
13
14 pragma solidity ^0.5.15;
15
16 import "openzeppelin-contracts/contracts/math/SafeMath.sol";
```

LOW Function parameter shadows a state variable.

The function parameter "owner" in contract "LoihiERC20" shadows the state variable with the same name "owner" in contract "LoihiRoot".

SWC-119 Source file

src/LoihiERC20.sol

Locations

```
136 * - `spender` cannot be the zero address.
137 */
138 function _approve(address owner, address spender, uint256 amount) internal {
139     require(owner != address(0), "ERC20: approve from the zero address");
140     require(spender != address(0), "ERC20: approve to the zero address");
```

LOW Loop over unbounded data structure.

Gas consumption in function "rpow" in contract "DSMath" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

lib/ds-math/src/math.sol

Locations

```
74 | z = n % 2 != 0 ? x : RAY;
75 |
76 | for (n /= 2; n != 0; n /= 2) {
77 |     x = rmul(x, x);
```

LOW Unused state variable "notEntered".

The state variable "notEntered" is declared within the contract "LoihiRoot" but its value does not seem to be used anywhere.

SWC-131

Source file

src/LoihiRoot.sol

Locations

```
32 |
33 | address public owner;
34 | bool internal notEntered = true;
35 | bool internal frozen = false;
```

LOW Unused state variable "frozen".

The state variable "frozen" is declared within the contract "LoihiRoot" but its value does not seem to be used anywhere.

SWC-131

Source file

src/LoihiRoot.sol

Locations

```
33 | address public owner;
34 | bool internal notEntered = true;
35 | bool internal frozen = false;
36 |
37 | uint256 alpha;
```

LOW Unused state variable "alpha".

The state variable "alpha" is declared within the contract "LoihiRoot" but its value does not seem to be used anywhere.

SWC-131

Source file

src/LoihiRoot.sol

Locations

```
35 | bool internal frozen = false;
36 |
37 | uint256 alpha;
38 | uint256 beta;
39 | uint256 feeBase;
```

LOW Unused state variable "beta".

The state variable "beta" is declared within the contract "LoihiRoot" but its value does not seem to be used anywhere.

SWC-131

Source file

src/LoihiRoot.sol

Locations

```
36 |
37 |     uint256 alpha;
38 |     uint256 beta;
39 |     uint256 feeBase;
40 |     uint256 feeDerivative;
```

LOW Unused state variable "feeBase".

The state variable "feeBase" is declared within the contract "LoihiRoot" but its value does not seem to be used anywhere.

SWC-131

Source file

src/LoihiRoot.sol

Locations

```
37 |     uint256 alpha;
38 |     uint256 beta;
39 |     uint256 feeBase;
40 |     uint256 feeDerivative;
```

LOW Unused state variable "feeDerivative".

The state variable "feeDerivative" is declared within the contract "LoihiRoot" but its value does not seem to be used anywhere.

SWC-131

Source file

src/LoihiRoot.sol

Locations

```
38 |     uint256 beta;
39 |     uint256 feeBase;
40 |     uint256 feeDerivative;
41 |
42 |     bytes4 constant internal ERC20ID = 0x36372b07;
```

Started Tue Jul 07 2020 00:32:02 GMT+0000 (Coordinated Universal Time)

Finished Tue Jul 07 2020 01:17:13 GMT+0000 (Coordinated Universal Time)

Mode Deep

Client Tool Mythx-Cli-0.6.19

Main Source File Src/LoihiExchange.sol

DETECTED VULNERABILITIES



ISSUES

MEDIUM Loop over unbounded data structure.

Gas consumption in function "getOriginTradeVariables" in contract "LoihiExchange" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

src/LoihiExchange.sol

Locations

```
79 | oBal_ = add(oBal_, oNAmt_);  
80 |  
81 | for (uint i = 0; i < reserves.length; i++) {  
82 |   if (reserves[i] != _o.reserve && reserves[i] != _t.reserve){  
83 |     grossLiq_ += dGetNumeraireBalance(reserves[i]);
```

MEDIUM Loop over unbounded data structure.

Gas consumption in function "getTargetTradeVariables" in contract "LoihiExchange" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

src/LoihiExchange.sol

Locations

```
238 | tBal_ = sub(tBal_, tNAmt_);  
239 |  
240 | for (uint i = 0; i < reserves.length; i++) {  
241 |   if (reserves[i] != _o.reserve && reserves[i] != _t.reserve) {  
242 |     grossLiq_ += dGetNumeraireBalance(reserves[i]);
```

LOW A floating pragma is set.

The current pragma Solidity directive is `""^0.5.15""`. 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

src/LoihiExchange.sol

Locations

```
12 // along with this program. If not, see <http://www.gnu.org/licenses/>.
13
14 pragma solidity ^0.5.15;
15
16 import "./LoihiRoot.sol";
```

LOW A control flow decision is made based on The block.timestamp environment variable.

The `block.timestamp` environment variable is used to determine a control flow decision. 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

src/LoihiExchange.sol

Locations

```
187 /// @param _recipient the address for where to send the target amount
188 function executeTargetTrade (address _origin, address _target, uint256 _maxOAmt, uint256 _tAmt, uint256 _deadline, address _recipient) external returns (uint256) {
189   require (_deadline >= now, "deadline has passed for this trade");
190
191   Flavor memory _o = flavors[_origin];
```

LOW Loop over unbounded data structure.

Gas consumption in function `"rpow"` in contract `"DSMath"` depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

lib/ds-math/src/math.sol

Locations

```
74   z = n % 2 != 0 ? x : RAY;
75
76   for (n /= 2; n != 0; n /= 2) {
77     x = rmul(x, x);
```

LOW

Unused function parameter "_deadline".

The value of the function parameter "_deadline" for the function "executeOriginTrade" of contract "LoihiExchange" does not seem to be used anywhere in "executeOriginTrade".

SWC-131

Source file

src/LoihiExchange.sol

Locations

```
28 | /// @param _recipient the address for where to send the resultant target amount
29 | /// @return tNAmt_ the target numeraire amount
30 | function executeOriginTrade (address _origin, address _target, uint256 _oAmt, uint256 _minTAmt, uint256 _deadline, address _recipient) external returns (uint256 {
31 |
32 |     Flavor memory _o = flavors[_origin]; // origin adapter + weight
```

Started Tue Jul 07 2020 00:32:02 GMT+0000 (Coordinated Universal Time)

Finished Tue Jul 07 2020 01:17:16 GMT+0000 (Coordinated Universal Time)

Mode Deep

Client Tool Mythx-Cli-0.6.19

Main Source File Src/LoihiliQuidity.sol

DETECTED VULNERABILITIES

(HIGH) (MEDIUM) (LOW)

1 6 5

ISSUES

HIGH The arithmetic operator can overflow.

It is possible to cause an integer overflow or underflow in the arithmetic operation.

SWC-101

Source file

lib/ds-math/src/math.sol

Locations

```
24 }
25 function mul(uint x, uint y) internal pure returns (uint z) {
26     require(y == 0 || (z = x * y) / y == x, "ds-math-mul-overflow");
27 }
```

MEDIUM Loop over unbounded data structure.

Gas consumption in function "getBalancesTokenAmountsAndWeights" in contract "LoihiliQuidity" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

src/LoihiliQuidity.sol

Locations

```
36 require(_f.adapter != address(0), "flavor not supported");
37
38 for (uint j = 0; j < reserves.length; j++) {
39     if (balances_[j] == 0) balances_[j] = dGetNumeraireBalance(reserves[j]);
40     if (reserves[j] == _f.reserve && _amts[i] > 0) {
```

MEDIUM Loop over unbounded data structure.

Gas consumption in function "calculateShellsToBurn" in contract "Loihiliquidity" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

src/Loihiliquidity.sol

Locations

```
182 | uint256 _numeraireShellsToBurn;
183 |
184 | for (uint i = 0; i < reserves.length; i++) {
185 |   if (_withdrawals[i] == 0) continue;
186 |   uint256 _withdrawal = _withdrawals[i];
```

MEDIUM Loop over unbounded data structure.

Gas consumption in function "proportionalDeposit" in contract "Loihiliquidity" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

src/Loihiliquidity.sol

Locations

```
236 | if (_totalSupply == 0) {
237 |
238 | for (uint i = 0; i < reserves.length; i++) {
239 |   Flavor memory _f = flavors[numeraires[i]];
240 |   _amounts[i] = dIntakeNumeraire(_f.adapter, wmul(_f.weight, _deposit));
```

MEDIUM Loop over unbounded data structure.

Gas consumption in function "proportionalDeposit" in contract "Loihiliquidity" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

src/Loihiliquidity.sol

Locations

```
249 | } else {
250 |
251 | for (uint i = 0; i < reserves.length; i++) {
252 |   Flavor memory _f = flavors[numeraires[i]];
253 |   _amounts[i] = wmul(_f.weight, _deposit);
```

MEDIUM Loop over unbounded data structure.

Gas consumption in function "proportionalDeposit" in contract "Loihiliquidity" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

src/Loihiliquidity.sol

Locations

```
259 _mint(msg.sender, shellsToMint_);  
260  
261 for (uint i = 0; i < reserves.length; i++) {  
262 Flavor memory d = flavors[numeraires[i]];  
263 _amounts[i] = dIntakeNumeraire(d.adapter, _amounts[i]);
```

MEDIUM Loop over unbounded data structure.

Gas consumption in function "proportionalWithdraw" in contract "Loihiliquidity" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

src/Loihiliquidity.sol

Locations

```
284  
285 uint256[] memory withdrawalAmts_ = new uint256[](reserves.length);  
286 for (uint i = 0; i < reserves.length; i++) {  
287 uint256 amount = dGetNumeraireBalance(reserves[i]);  
288 uint256 proportionateValue = wmul(wmul(amount, _withdrawMultiplier), WAD - feeBase);
```

LOW A floating pragma is set.

The current pragma Solidity directive is ""^0.5.15"". 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

src/Loihiliquidity.sol

Locations

```
12 // along with this program. If not, see <http://www.gnu.org/licenses/>.  
13  
14 pragma solidity ^0.5.15;  
15  
16 import "./LoihirRoot.sol";
```

LOW

An assertion violation was triggered.

SWC-110

It is possible to cause an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source file

lib/ds-math/src/math.sol

Locations

```
44
45 function wmul(uint x, uint y) internal pure returns (uint z) {
46     z = addmul(x, y, WAD / 2) / WAD;
47 }
48 function rmul(uint x, uint y) internal pure returns (uint z) {
```

LOW

A control flow decision is made based on The block.timestamp environment variable.

SWC-116

The block.timestamp environment variable is used to determine a control flow decision. 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

src/Loihiliquidity.sol

Locations

```
56 /// @return shellsToMint_ the amount of shells to mint for the deposited stablecoin flavors
57 function selectiveDeposit (address[] calldata _flvrs, uint256[] calldata _amps, uint256 _minShells, uint256 _deadline) external returns (uint256 shellsToMint_) {
58     require(_deadline >= now, "deadline has passed for this transaction");
59
60     ( uint256[] memory _balances,
```

LOW

A control flow decision is made based on The block.timestamp environment variable.

SWC-116

The block.timestamp environment variable is used to determine a control flow decision. 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

src/Loihiliquidity.sol

Locations

```
143 /// @return shellsBurned_ the corresponding amount of shell tokens to withdraw the specified amount of specified flavors
144 function selectiveWithdraw (address[] calldata _flvrs, uint256[] calldata _amps, uint256 _maxShells, uint256 _deadline) external returns (uint256 shellsBurned_) {
145     require(_deadline >= now, "deadline has passed for this transaction");
146
147     ( uint256[] memory _balances,
```

LOW

Loop over unbounded data structure.

Gas consumption in function "rpow" in contract "DSMath" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

lib/ds-math/src/math.sol

Locations

```
74 | z = n % 2 != 0 ? x : RAY;
75 |
76 | for (n /= 2; n != 0; n /= 2) {
77 |     x = rmul(x, x);
```

Started	Tue Jul 07 2020 00:32:02 GMT+0000 (Coordinated Universal Time)
Finished	Tue Jul 07 2020 01:17:14 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.19
Main Source File	Src/LoihiViews.sol

DETECTED VULNERABILITIES

(HIGH) (MEDIUM) (LOW)

20 0 7

ISSUES

HIGH The arithmetic operator can overflow.

It is possible to cause an integer overflow or underflow in the arithmetic operation.

SWC-101

Source file

src/LoihiViews.sol

Locations

```
50 | function calculateOriginTradeOriginAmount (uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase, uint256
51 | _feeDerivative) external view returns (uint256) {
52 |
53 | require(_oBal <= wmul(_oWeight, wmul(_grossLiq, _alpha)), "origin swap origin halt check");
54 |
55 | uint256 oNAmt_;
```

HIGH The arithmetic operation can underflow.

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

src/LoihiViews.sol

Locations

```
169 | require(_tBal >= wmul(_tWeight, wmul(_grossLiq, WAD - _alpha)), "target halt check for target trade");
170 |
171 | uint256 _feeThreshold = wmul(_tWeight, wmul(_grossLiq, WAD - _beta));
172 | if (_tBal >= _feeThreshold) {
```

HIGH

The arithmetic operation can underflow.

SWC-101

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

src/LoihiViews.sol

Locations

```
167 | function calculateTargetTradeTargetAmount(uint256 _tWeight, uint256 _tBal, uint256 _tNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase, uint256
168 | _feeDerivative) external view returns (uint256 tNAmt_) {
169 |
170 | require(_tBal >= wmul(_tWeight, wmul(_grossLiq, WAD - _alpha)), "target halt check for target trade");
171 |
172 | uint256 _feeThreshold = wmul(_tWeight, wmul(_grossLiq, WAD - _beta));
```

HIGH

The arithmetic operation can underflow.

SWC-101

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

src/LoihiViews.sol

Locations

```
130 | }
131 |
132 | require(sub(_tBal, tNAmt_) >= wmul(_tWeight, wmul(_grossLiq, WAD - _alpha)), "origin swap target halt check");
133 |
134 | return dViewRawAmount(_tAdptr, tNAmt_);
```

HIGH

The arithmetic operation can underflow.

SWC-101

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

src/LoihiViews.sol

Locations

```
100 | if (sub(_tBal, _tNAmt) >= _feeThreshold) {
101 |
102 | tNAmt_ = wmul(_tNAmt, WAD - _feeBase);
103 |
104 | } else if (_tBal <= _feeThreshold) {
```

HIGH

The arithmetic operation can underflow.

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

src/LoihiViews.sol

Locations

```
110 | _fee = wmul(_fee, _feeDerivative);
111 | _tNAmt = wmul(_tNAmt, WAD - _fee);
112 | tNAmt_ = wmul(_tNAmt, WAD - _feeBase);
113 |
114 } else {
```

HIGH

The arithmetic operation can underflow.

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

src/LoihiViews.sol

Locations

```
96 | function calculateOriginTradeTargetAmount (address _tAdptr, uint256 _tWeight, uint256 _tBal, uint256 _tNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase,
97 | uint256 _feeDerivative) external view returns (uint256 tNAmt_) {
98 |
99 | uint256 _feeThreshold = wmul(_tWeight, wmul(_grossLiq, WAD - _beta));
100 |
101 | if (sub(_tBal, _tNAmt) >= _feeThreshold) {
```

HIGH

The arithmetic operation can overflow.

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

src/LoihiViews.sol

Locations

```
26 | viewVars[1] = dViewNumeraireBalance(_oAdptr, _this);
27 | viewVars[3] += viewVars[1];
28 | viewVars[1] += viewVars[0];
29 |
30 | viewVars[2] = dViewNumeraireBalance(_tAdptr, _this);
```

HIGH

The arithmetic operation can underflow.

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

src/LoihiViews.sol

Locations

```
143 | viewVars[1] = dViewNumeraireBalance(_tAdptr, _this);
144 | viewVars[3] += viewVars[1];
145 | viewVars[1] -= viewVars[0];
146 |
147 | viewVars[2] = dViewNumeraireBalance(_oAdptr, _this);
```

HIGH

The arithmetic operation can underflow.

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

src/LoihiViews.sol

Locations

```
66 | );
67 | _fee = wmul(_fee, _feeDerivative);
68 | oNAmt_ = wmul(_oNAmt, WAD - _fee);
69 |
70 | } else {
```

HIGH

The arithmetic operation can underflow.

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

src/LoihiViews.sol

Locations

```
109 | );
110 | _fee = wmul(_fee, _feeDerivative);
111 | _tNAmt = wmul(_tNAmt, WAD - _fee);
112 | tNAmt_ = wmul(_tNAmt, WAD - _feeBase);
```

HIGH

The arithmetic operation can overflow.

SWC-101

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

src/LoihiViews.sol

Locations

```
29 | 
30 | viewVars[2] = dViewNumeraireBalance(_tAdptr, _this);
31 | viewVars[3] += viewVars[2];
32 | 
33 | for (uint i = 0; i < _rsrvs.length; i++) {
```

HIGH

The arithmetic operation can overflow.

SWC-101

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

src/LoihiViews.sol

Locations

```
33 | for (uint i = 0; i < _rsrvs.length; i++) {
34 | if (_rsrvs[i] != _oRsrv && _rsrvs[i] != _tRsrv) {
35 | viewVars[3] += dViewNumeraireBalance(_rsrvs[i], _this);
36 | }
37 | }
```

HIGH

The arithmetic operation can overflow.

SWC-101

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

src/LoihiViews.sol

Locations

```
146 | 
147 | viewVars[2] = dViewNumeraireBalance(_oAdptr, _this);
148 | viewVars[3] += viewVars[2];
149 | 
150 | for (uint i = 0; i < _rsrvs.length; i++) {
```

HIGH The arithmetic operation can overflow.

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

src/LoihiViews.sol

Locations

```
150 | for (uint i = 0; i < _rsrvs.length; i++) {  
151 |     if (_rsrvs[i] != _oRsrv && _rsrvs[i] != _tRsrv) {  
152 |         viewVars[3] += dViewNumeraireBalance(_rsrvs[i], _this);  
153 |     }  
154 | }
```

HIGH The arithmetic operation can overflow.

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

src/LoihiViews.sol

Locations

```
249 | for (uint i = 0; i < _reserves.length; i++) {  
250 |     balances[i] = dViewNumeraireBalance(_reserves[i], _addr);  
251 |     totalBalance += balances[i];  
252 | }  
253 | return (totalBalance, balances);
```

HIGH The arithmetic operation can underflow.

It is possible to cause an arithmetic underflow. Prevent the underflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the underflow.

Source file

src/LoihiViews.sol

Locations

```
79 | oNAmt_ = add(  
80 |     sub(_feeThreshold, sub(_oBal, _oNAmt)),  
81 |     wmul(sub(_oBal, _feeThreshold), WAD - _fee)  
82 | );
```

HIGH The arithmetic operator can overflow.

It is possible to cause an integer overflow or underflow in the arithmetic operation.

SWC-101

Source file

src/LoihiViews.sol

Locations

```
54 | uint256 oNAmt_;  
55 |  
56 |     uint256 _feeThreshold = wmul(_oWeight, wmul(_grossLiq, [beta] + WAD));  
57 |     if (_oBal <= _feeThreshold) {
```

HIGH

The arithmetic operation can overflow.

SWC-101

Source file

src/LoihiViews.sol

Locations

```
210 | function calculateTargetTradeOriginAmount (address _oAdptr, uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase,
211 | uint256 _feeDerivative) external view returns (uint256 oNAmt_) {
212 |
213 |     uint256 _feeThreshold = wmul(_oWeight, wmul(_grossLiq, WAD+_beta));
|     if (_oBal + _oNAmt <= _feeThreshold) {
```

HIGH

The arithmetic operation can overflow.

SWC-101

Source file

src/LoihiViews.sol

Locations

```
172 | if (_tBal >= _feeThreshold) {
173 |
174 |     tNAmt_ = wmul(_tNAmt, WAD+_feeBase);
175 |
176 | } else if (add(_tBal, _tNAmt) <= _feeThreshold) {
```

LOW

A floating pragma is set.

SWC-103

Source file

src/LoihiViews.sol

Locations

```
12 | // along with this program. If not, see <http://www.gnu.org/licenses/>.
13 |
14 | pragma solidity ^0.5.15;
15 |
16 | import "./LoihiRoot.sol";
```

LOW

An assertion violation was triggered.

SWC-110

It is possible to cause an assertion violation. Note that Solidity assert() statements should only be used to check invariants. Review the transaction trace generated for this issue and either make sure your program logic is correct, or use require() instead of assert() if your goal is to constrain user inputs or enforce preconditions. Remember to validate inputs from both callers (for instance, via passed arguments) and callees (for instance, via return values).

Source file

lib/ds-math/src/math.sol

Locations

```
44 |
45 | function wmul(uint x, uint y) internal pure returns (uint z) {
46 |     z = addmul(x, y, WAD / 2) / WAD;
47 |
48 |     function rmul(uint x, uint y) internal pure returns (uint z) {
```

LOW

Local variable shadows a state variable.

The local variable "balances" in contract "LoihiViews" shadows the state variable with the same name "balances" in contract "LoihiRoot".

SWC-119

Source file

src/LoihiViews.sol

Locations

```
246 | function totalReserves(address[] calldata _reserves, address _addr) external view returns (uint256, uint256[] memory) {
247 |     uint256 totalBalance;
248 |     uint256[] memory balances = new uint256[](_reserves.length);
249 |     for (uint i = 0; i < _reserves.length; i++) {
250 |         balances[i] = dViewNumeraireBalance(_reserves[i], _addr);
```

LOW

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

SWC-123

Source file

src/LoihiDelegators.sol

Locations

```
27 | 
28 | function staticTo(address callee, bytes memory data) internal view returns (bytes memory) {
29 |     (bool success, bytes memory returnData) = callee.staticcall(data);
30 |     assembly {
31 |         if eq(success, 0) {
```

Source file

src/LoihiViews.sol

Locations

```
17 | import "./LoihiDelegators.sol";
18 |
19 | contract LoihiViews is LoihiRoot, LoihiDelegators {
20 |
21 |     function getOriginViewVariables (address _this, address[] calldata _rsrvs, address _oAdptr, address _oRsrv, address _tAdptr, address _tRsrv, uint256 _oAmt) external view returns
22 |     (uint256[] memory) {
23 |
24 |         uint256[] memory viewVars = new uint256[](4);
25 |
26 |         viewVars[0] = dViewNumeraireAmount(_oAdptr, _oAmt);
27 |         viewVars[1] = dViewNumeraireBalance(_oAdptr, _this);
28 |         viewVars[3] += viewVars[1];
29 |         viewVars[1] += viewVars[0];
30 |
31 |         viewVars[2] = dViewNumeraireBalance(_tAdptr, _this);
32 |         viewVars[3] += viewVars[2];
33 |
34 |         for (uint i = 0; i < _rsrvs.length; i++) {
35 |             if (_rsrvs[i] != _oRsrv && _rsrvs[i] != _tRsrv) {
36 |                 viewVars[3] += dViewNumeraireBalance(_rsrvs[i], _this);
37 |             }
38 |         }
39 |
40 |         return viewVars;
41 |
42 |     }
43 |
44 |     /// @author james foley http://github.com/realisation
45 |     /// @notice calculates the origin amount in an origin trade including the fees
46 |     /// @param _oWeight the balance weighting of the origin flavor
47 |     /// @param _oBal the new numeraire balance of the origin reserve including the origin amount being swapped
48 |     /// @param _oNAmt the origin numeraire amount being swapped
49 |     /// @param _grossLiq the numeraire amount across all stablecoin reserves in the contract
50 |     /// @return _oNAmt_ the origin numeraire amount for the swap with fees applied
51 |     function calculateOriginTradeOriginAmount (uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase, uint256
52 |     _feeDerivative) external view returns (uint256) {
53 |
54 |         require(_oBal <= _mul(_oWeight, _mul(_grossLiq, _alpha + WAD)), "origin swap origin halt check");
55 |
56 |         uint256 _oNAmt_ = _oNAmt;
57 |
58 |         uint256 _feeThreshold = _mul(_oWeight, _mul(_grossLiq, _beta + WAD));
59 |         if (_oBal <= _feeThreshold) {
60 |
61 |             _oNAmt_ = _oNAmt;
```

```

62
63 ) else if (_oBal < _oNAmt) >= _feeThreshold) {
64
65     uint256 _fee = wdiv(
66         sub(_oBal, _feeThreshold),
67         mmul(_oWeight, _grossLiq),
68     );
69     _fee = wmul(_fee, _feeDerivative);
70     oNAmt_ = wmul(_oNAmt, WAD - _fee);
71
72 } else {
73
74     uint256 _fee = wdiv(
75         sub(_oBal, _feeThreshold),
76         mmul(_oWeight, _grossLiq),
77     );
78
79     _fee = wmul(_feeDerivative, _fee);
80
81     oNAmt_ = add(
82         sub(_feeThreshold, sub(_oBal, _oNAmt)),
83         mmul(sub(_oBal, _feeThreshold), WAD - _fee)
84     );
85
86 }
87
88 return oNAmt_;
89
90
91
92 /// @author james foley http://github.com/realisation
93 /// @notice calculates the fees to apply to the target amount in an origin trade
94 /// @param _tWeight the balance weighting of the target flavor
95 /// @param _tBal the current balance of the target in the reserve
96 /// @param _grossLiq the current total balance across all the reserves in the contract
97 /// @return tNAmt_ the target numeraire amount including any applied fees
98 function calculateOriginTradeTargetAmount (address _tAdptr, uint256 _tWeight, uint256 _tBal, uint256 _tNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase,
99     uint256 _feeDerivative) external view returns (uint256 tNAmt_) {
100
101     uint256 _feeThreshold = mmul(_tWeight, mmul(_grossLiq, WAD - _beta));
102
103     if (sub(_tBal, _tNAmt) >= _feeThreshold) {
104
105         tNAmt_ = wmul(_tNAmt, WAD - _feeBase);
106
107     } else if (_tBal <= _feeThreshold) {
108
109         uint256 _fee = wdiv(
110             sub(_feeThreshold, sub(_tBal, _tNAmt)),
111             mmul(_tWeight, _grossLiq),
112         );
113         _fee = wmul(_fee, _feeDerivative);
114         _tNAmt = wmul(_tNAmt, WAD - _fee);
115         tNAmt_ = wmul(_tNAmt, WAD - _feeBase);
116
117     } else {
118
119         uint256 _fee = wdiv(
120             sub(_feeThreshold, sub(_tBal, _tNAmt)),
121             mmul(_tWeight, _grossLiq),
122         );
123
124         _fee = wmul(_feeDerivative, _fee);

```

```

125
126 tNAmt_ = add(
127 sub(_tBal, _feeThreshold));
128 wmul(sub(_feeThreshold, sub(_tBal, _tNAmt)), WAD) - _fee
129 );
130
131 tNAmt_ = wmul(tNAmt_, WAD) - _feeBase);
132
133 )
134
135 require(sub(_tBal, tNAmt_) >= wmul(_tWeight, wmul(_grossLiq, WAD) - _alpha), "origin swap target halt check");
136
137 return dViewRawAmount(_tAdptr, tNAmt_);
138
139 )
140
141 function getTargetViewVariables (address _this, address[] calldata _rsrvs, address _oAdptr, address _oRsrv, address _tAdptr, address _tRsrv, uint256 _tAmt) external view returns
142 (uint256[] memory);
143
144 uint256[] memory viewVars = new uint256[](4);
145
146 viewVars[0] = dViewNumeraireAmount(_tAdptr, _tAmt);
147 viewVars[1] = dViewNumeraireBalance(_tAdptr, _this);
148 viewVars[3] += viewVars[1];
149 viewVars[1] -= viewVars[0];
150
151 viewVars[2] = dViewNumeraireBalance(_oAdptr, _this);
152 viewVars[3] += viewVars[2];
153
154 for (uint i = 0; i < _rsrvs.length; i++) {
155 if (_rsrvs[i] != _oRsrv && _rsrvs[i] != _tRsrv) {
156 viewVars[3] += dViewNumeraireBalance(_rsrvs[i], _this);
157 }
158 }
159
160 return viewVars;
161
162
163
164 /// @author james foley http://github.com/realisation
165 /// @notice this function applies fees to the target amount according to how balanced it is relative to its weight
166 /// @param _tWeight the weighted balance point of the target token
167 /// @param _tBal the contract's balance of the target
168 /// @param _tNAmt the numeraire value of the target amount being traded
169 /// @param _grossLiq the total numeraire value of all liquidity across all the reserves of the contract
170 /// @return tNAmt_ the target numeraire amount after applying fees
171 function calculateTargetTradeTargetAmount(uint256 _tWeight, uint256 _tBal, uint256 _tNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase, uint256
172 _feeDerivative) external view returns (uint256 tNAmt_);
173
174 require(_tBal >= wmul(_tWeight, wmul(_grossLiq, WAD) - _alpha), "target halt check for target trade");
175
176 uint256 _feeThreshold = wmul(_tWeight, wmul(_grossLiq, WAD) - _beta);
177 if (_tBal >=_feeThreshold) {
178
179 tNAmt_ = wmul(_tNAmt, WAD + _feeBase);
180
181 } else if (add(_tBal, _tNAmt) <= _feeThreshold) {
182
183 uint256 _fee = mdv(sub(_feeThreshold, _tBal), wmul(_tWeight, _grossLiq));
184 _fee = wmul(_fee, _feeDerivative);
185 _tNAmt = wmul(_tNAmt, WAD + _fee);
186 tNAmt_ = wmul(_tNAmt, WAD + _feeBase);
187

```

```

188 } else {
189
190     uint256 _fee = wmul(_feeDerivative, wdiv(
191         sub(_feeThreshold, _tBal),
192         mul(_tWeight, _grossLiq)
193     ));
194
195     _tNAmt = add(
196         sub(add(_tBal, _tNAmt), _feeThreshold),
197         mul(sub(_feeThreshold, _tBal), WAD + _fee)
198     );
199
200     tNAmt_ = wmul(_tNAmt, WAD + _feeBase);
201
202 }
203
204 return tNAmt_;
205
206 }
207
208 /**
209  * @author james foley http://github.com/realisation
210  *
211  * @notice this function applies fees to the origin amount according to how balanced it is relative to its weight
212  * @param _oWeight the weighted balance point of the origin token
213  * @param _oBal the contract's balance of the origin
214  * @param _oNAmt the numeraire value for the origin amount being traded
215  * @param _grossLiq the total numeraire value of all liquidity across all the reserves of the contract
216  * @return _oNAmt_ the origin numeraire amount after applying fees
217  */
218 function calculateTargetTradeOriginAmount (address _oAdptr, uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase,
219     uint256 _feeDerivative) external view returns (uint256 _oNAmt_) {
220
221     uint256 _feeThreshold = wmul(_oWeight, wmul(_grossLiq, WAD + _beta));
222
223     if (_oBal + _oNAmt_ <= _feeThreshold) {
224
225         _oNAmt_ = _oNAmt;
226
227     } else if (_oBal >= _feeThreshold) {
228
229         uint256 _fee = wdiv(
230             sub(add(_oNAmt, _oBal), _feeThreshold),
231             mul(_oWeight, _grossLiq)
232         );
233
234         _fee = wmul(_fee, _feeDerivative);
235
236         _oNAmt_ = wmul(_oNAmt, WAD + _fee);
237
238     } else {
239
240         uint256 _fee = wmul(_feeDerivative, wdiv(
241             sub(add(_oBal, _oNAmt), _feeThreshold),
242             mul(_oWeight, _grossLiq)
243         ));
244
245         _oNAmt_ = add(
246             sub(_feeThreshold, _oBal),
247             mul(sub(add(_oBal, _oNAmt), _feeThreshold), WAD + _fee)
248         );
249
250     }
251
252     require(add(_oBal, _oNAmt_) <= wmul(_oWeight, wmul(_grossLiq, WAD + _alpha)), "origin halt check for target trade");
253
254     return dViewRawAmount(_oAdptr, _oNAmt_);
255 }

```

```

251
252 function totalReserves (address[] calldata _reserves, address _addr external view returns (uint256, uint256[] memory) {
253     uint256 totalBalance;
254     uint256[] memory balances = new uint256[](_reserves.length);
255     for (uint i = 0; i < _reserves.length; i++) {
256         balances[i] = dViewNumeraireBalance(_reserves[i], _addr);
257         totalBalance += balances[i];
258     }
259     return (totalBalance, balances);
260 }
```

LOW Loop over unbounded data structure.

Gas consumption in function "pow" in contract "DSMath" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

`lib/ds-math/src/math.sol`

Locations

```

74 | z = n % 2 != 0 ? x : RAY;
75 |
76 | for (n /= 2; n != 0; n /= 2) {
77 |     x = rmul(x, x);
```

LOW Unused function parameter "_feeBase".

The value of the function parameter "_feeBase" for the function "calculateOriginTradeOriginAmount" of contract "LoihiViews" does not seem to be used anywhere in "calculateOriginTradeOriginAmount".

SWC-131

Source file

`src/LoihiViews.sol`

Locations

```

48 | /// @param _grossLiq the numeraire amount across all stablecoin reserves in the contract
49 | /// @return oNAmt_ the origin numeraire amount for the swap with fees applied
50 | function calculateOriginTradeOriginAmount (uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase, uint256
51 | _feeDerivative) external view returns (uint256) {
52 |
53 |     require(_oBal <= wmul(_oWeight, wmul(_grossLiq, _alpha + WAD)), "origin swap origin halt check");
```

LOW

Unused function parameter "_feeBase".

The value of the function parameter "_feeBase" for the function "calculateTargetTradeOriginAmount" of contract "LoihiViews" does not seem to be used anywhere in "calculateTargetTradeOriginAmount".

SWC-131

Source file

src/LoihiViews.sol

Locations

```
208 | /// @param _grossLiq the total numeraire value of all liquidity across all the reserves of the contract
209 | /// @return oNAmt_ the origin numeraire amount after applying fees
210 | function calculateTargetTradeOriginAmount (address _oAdptr, uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase,
211 | uint256 _feeDerivative) external view returns (uint256 oNAmt_) {
212 |
  uint256 _feeThreshold = wmul(_oWeight, wmul(_grossLiq, WAD + _beta));
```