MythX

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

Analysis 6df956e5-3745-4600-ba16-734b214c8d16 MythX

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

HIGH	MEDIUM	LOW
0	0	3

ISSUES

LOW

SWC-103

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/Loihi.sol

Locations

```
// along with this program. If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.15;

import "./LoihiRoot.sol";
```

LOW

SWC-119

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".

Source file

src/Loihi.sol

Locations

```
}

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

LOW

SWC-131

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".

Source file

`src/Loihi.sol`

Locations

```

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

Analysis e3933c9e-1e8c-4fa8-bac4-3886c1b5d906

MythX

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

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

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/LoihiDelegators.sol

Locations

```
// along with this program. If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.15;

contract LoihiDelegators {
```

Analysis 6147bc39-5f33-4e3a-8b2e-189d40b5e990 MythX

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

HIGH	MEDIUM	LOW
0	0	9

ISSUES

LOW SWC-103 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/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 SWC-119 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".

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

SWC-128

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

```
z = n % 2 != 0 ? x : RAY;

for (n /= 2; n != 0; n /= 2) {
x = rmul(x, x);
```

LOW

SWC-131

Unused state variable "notEntered".

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

Source file

`src/LoihiRoot.sol`

Locations

```

address public owner;
bool internal notEntered = true;
bool internal frozen = false;
```

LOW

SWC-131

Unused state variable "frozen".

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

Source file

`src/LoihiRoot.sol`

Locations

```
address public owner;
bool internal notEntered = true;
bool internal frozen = false;

uint256 alpha;
```

LOW

SWC-131

Unused state variable "alpha".

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

Source file

`src/LoihiRoot.sol`

Locations

```
bool internal frozen = false;

uint256 alpha;
uint256 beta;
uint256 feeBase;
```

LOW

SWC-131

Unused state variable "beta".

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

Source file

`src/LoihiRoot.sol`

Locations

```

uint256 alpha;
uint256 beta;
uint256 feeBase;
uint256 feeDerivative;
```

LOW

SWC-131

Unused state variable "feeBase".

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

Source file

`src/LoihiRoot.sol`

Locations

```
uint256 alpha;
uint256 beta;
uint256 feeBase;
uint256 feeDerivative;
```

LOW

SWC-131

Unused state variable "feeDerivative".

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

Source file

`src/LoihiRoot.sol`

Locations

```
uint256 beta;
uint256 feeBase;
uint256 feeDerivative;

bytes4 constant internal ERC20ID = 0x36372b07;
```

Analysis a26303f1-eb49-420f-ae51-5728493c1e06

MythX

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

HIGH	MEDIUM	LOW
0	2	4

ISSUES

MEDIUM SWC-128 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

```
oBal_ = add(oBal_, oNAmt_);

for (uint i = 0; i < reserves.length; i++) {
if (reserves[i] != _o.reserve && reserves[i] != _t.reserve){
grossLiq_ += dGetNumeraireBalance(reserves[i]);
```

MEDIUM SWC-128 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

```
tBal_ = sub(tBal_, tNAmt_);

for (uint i = 0; i < reserves.length; i++) {
if (reserves[i] != _o.reserve && reserves[i] != _t.reserve) {
grossLiq_ += dGetNumeraireBalance(reserves[i]);
```

LOW SWC-103

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

```
// along with this program. If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.15;

import "./LoihiRoot.sol";
```

LOW SWC-116

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

```
/// @param _recipient the address for where to send the target amount
function executeTargetTrade (address _origin, address _target, uint256 _maxOAmt, uint256 _tAmt, uint256 _deadline, address _recipient) external returns (uint256) {
require(_deadline >= now, "deadline has passed for this trade");

Flavor memory _o = flavors[_origin];
```

LOW SWC-128

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

```
z = n % 2 != 0 ? x : RAY;

for (n /= 2; n != 0; n /= 2) {
x = rmul(x, x);
```

LOW

SWC-131

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".

Source file

`src/LoihiExchange.sol`

Locations

```
/// @param _recipient the address for where to send the resultant target amount
/// @return tNAmt_ the target numeraire amount
function executeOriginTrade (address _origin, address _target, uint256 _oAmt, uint256 _minTAmt, uint256 _deadline, address _recipient) external returns (uint256) {

Flavor memory _o = flavors[_origin]; // origin adapter + weight
```

Analysis 94aef017-b727-4d0e-b9b8-33e3477b9e1f

MythX

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 SWC-101

The arithmetic operator can overflow.

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

Source file

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

Locations

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

MEDIUM SWC-128

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

```
require(_f.adapter != address(0), "flavor not supported");

for (uint j = 0; j < reserves.length; j++) {
if (balances_[j] == 0) balances_[j] = dGetNumeraireBalance(reserves[j]);
if (reserves[j] == _f.reserve && _amts[i] > 0) {
```

MEDIUM

SWC-128

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

```
uint256 _numeraireShellsToBurn;

for (uint i = 0; i < reserves.length; i++) {
if (_withdrawals[i] == 0) continue;
uint256 _withdrawal = _withdrawals[i];
```

MEDIUM

SWC-128

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

```
if (_totalSupply == 0) {

for (uint i = 0; i < reserves.length; i++) {
Flavor memory _f = flavors[numeraires[i]];
_amounts[i] = dIntakeNumeraire(_f.adapter, wmul(_f.weight, _deposit));
```

MEDIUM

SWC-128

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

```
} else {

for (uint i = 0; i < reserves.length; i++) {
Flavor memory _f = flavors[numeraires[i]];
_amounts[i] = wmul(_f.weight, _deposit);
```

MEDIUM

SWC-128

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

```
_mint(msg.sender, shellsToMint_);

for (uint i = 0; i < reserves.length; i++) {
Flavor memory d = flavors[numeraires[i]];
_amounts[i] = dIntakeNumeraire(d.adapter, _amounts[i]);
```

MEDIUM

SWC-128

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

```

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

LOW

SWC-103

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

```
// along with this program. If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.15;

import "./LoihiRoot.sol";
```

LOW

SWC-110

An assertion violation was triggered.

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

```

function wmul(uint x, uint y) internal pure returns (uint z) {
z = add(mul(x, y), WAD / 2) / WAD;
}
function rmul(uint x, uint y) internal pure returns (uint z) {
```

LOW

SWC-116

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/LoihiLiquidity.sol`

Locations

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

( uint256[] memory _balances,
```

LOW

SWC-116

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/LoihiLiquidity.sol`

Locations

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

( uint256[] memory _balances,
```

LOW

SWC-128

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);
```

Analysis 7f8a2fa5-d6d7-4388-bc74-bf273a7a3f9e

MythX

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 SWC-101

The arithmetic operator can overflow.

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

Source file

src/LoihiViews.sol

Locations

```
function calculateOriginTradeOriginAmount (uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase, uint256
_feeDerivative) external view returns (uint256) {

require(_oBal <= wmul(_oWeight, wmul(_grossLiq, _alpha + WAD)), "origin swap origin halt check");

uint256 oNAmt_;
```

HIGH SWC-101

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

```
require(_tBal >= wmul(_tWeight, wmul(_grossLiq, WAD - _alpha)), "target halt check for target trade");

uint256 _feeThreshold = wmul(_tWeight, wmul(_grossLiq, WAD - _beta));
if (_tBal >= _feeThreshold) {
```

HIGH

SWC-101

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

```
function calculateTargetTradeTargetAmount(uint256 _tWeight, uint256 _tBal, uint256 _tNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase, uint256
_feeDerivative) external view returns (uint256 tNAmt_) {

require(_tBal >= wmul(_tWeight, wmul(_grossLiq, WAD - _alpha)), "target halt check for target trade");

uint256 _feeThreshold = wmul(_tWeight, wmul(_grossLiq, WAD - _beta));
```

HIGH

SWC-101

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

```
}

require(sub(_tBal, tNAmt_) >= wmul(_tWeight, wmul(_grossLiq, WAD - _alpha)), "origin swap target halt check");

return dViewRawAmount(_tAdptr, tNAmt_);
```

HIGH

SWC-101

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

```
if (sub(_tBal, _tNAmt) >= _feeThreshold) {

tNAmt_ = wmul(_tNAmt, WAD - _feeBase);

} else if (_tBal <= _feeThreshold) {
```

HIGH

SWC-101

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

```
_fee = wmul(_fee, _feeDerivative);
_tNAmt = wmul(_tNAmt, WAD - _fee);
tNAmt_ = wmul(_tNAmt, WAD - _feeBase);

} else {
```

HIGH

SWC-101

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

```
function calculateOriginTradeTargetAmount (address _tAdptr, uint256 _tWeight, uint256 _tBal, uint256 _tNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase,
uint256 _feeDerivative) external view returns (uint256 tNAmt_) {

uint256 _feeThreshold = wmul(_tWeight, wmul(_grossLiq, WAD - _beta));

if (sub(_tBal, _tNAmt) >= _feeThreshold) {
```

HIGH

SWC-101

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

```
viewVars[1] = dViewNumeraireBalance(_oAdptr, _this);
viewVars[3] += viewVars[1];
viewVars[1] += viewVars[0];

viewVars[2] = dViewNumeraireBalance(_tAdptr, _this);
```

HIGH

SWC-101

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

```
viewVars[1] = dViewNumeraireBalance(_tAdptr, _this);
viewVars[3] += viewVars[1];
viewVars[1] -= viewVars[0];

viewVars[2] = dViewNumeraireBalance(_oAdptr, _this);
```

HIGH

SWC-101

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

```
);
_fee = wmul(_fee, _feeDerivative);
oNAmt_ = wmul(_oNAmt, WAD - _fee);

} else {
```

HIGH

SWC-101

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

```
);
_fee = wmul(_fee, _feeDerivative);
_tNAmt = wmul(_tNAmt, WAD - _fee);
tNAmt_ = wmul(_tNAmt, WAD - _feeBase);
```

HIGH

SWC-101

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

```

viewVars[2] = dViewNumeraireBalance(_tAdptr, _this);
viewVars[3] += viewVars[2];

for (uint i = 0; i < _rsrvs.length; i++) {
```

HIGH

SWC-101

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

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

HIGH

SWC-101

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

```

viewVars[2] = dViewNumeraireBalance(_oAdptr, _this);
viewVars[3] += viewVars[2];

for (uint i = 0; i < _rsrvs.length; i++) {
```

HIGH

SWC-101

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

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

HIGH

SWC-101

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

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

HIGH

SWC-101

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

```
oNAmt_ = add(
sub(_feeThreshold, sub(_oBal, _oNAmt)),
wmul(sub(_oBal, _feeThreshold), WAD - _fee)
);
```

HIGH

SWC-101

The arithmetic operator can overflow.

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

Source file

src/LoihiViews.sol

Locations

```
uint256 oNAmt_;

uint256 _feeThreshold = wmul(_oWeight, wmul(_grossLiq, _beta + WAD));
if (_oBal <= _feeThreshold) {
```

HIGH

SWC-101

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

```
function calculateTargetTradeOriginAmount (address _oAdptr, uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase,
uint256 _feeDerivative) external view returns (uint256 oNAmt_) {

uint256 _feeThreshold = wmul(_oWeight, wmul(_grossLiq, WAD + _beta));
if (_oBal + _oNAmt <= _feeThreshold) {
```

HIGH

SWC-101

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

```
if (_tBal >= _feeThreshold) {

tNAmt_ = wmul(_tNAmt, WAD + _feeBase);

} else if (add(_tBal, _tNAmt) <= _feeThreshold) {
```

LOW

SWC-103

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/LoihiViews.sol`

Locations

```
// along with this program. If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.15;

import "./LoihiRoot.sol";
```

LOW

SWC-110

An assertion violation was triggered.

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

```

function wmul(uint x, uint y) internal pure returns (uint z) {
z = add(mul(x, y), WAD / 2) / WAD;
}
function rmul(uint x, uint y) internal pure returns (uint z) {
```

LOW

SWC-119

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".

Source file

`src/LoihiViews.sol`

Locations

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

LOW

SWC-123

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).

Source file

src/LoihiDelegators.sol

Locations

```

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

Source file

src/LoihiViews.sol

Locations

```
import "./LoihiDelegators.sol";

contract LoihiViews is LoihiRoot, LoihiDelegators {

function getOriginViewVariables (address _this, address[] calldata _rsrvs, address _oAdptr, address _oRsrv, address _tAdptr, address _tRsrv, uint256 _oAmt) external view returns
(uint256[] memory) {

uint256[] memory viewVars = new uint256[](4);

viewVars[0] = dViewNumeraireAmount(_oAdptr, _oAmt);
viewVars[1] = dViewNumeraireBalance(_oAdptr, _this);
viewVars[3] += viewVars[1];
viewVars[1] += viewVars[0];

viewVars[2] = dViewNumeraireBalance(_tAdptr, _this);
viewVars[3] += viewVars[2];

for (uint i = 0; i < _rsrvs.length; i++) {
if (_rsrvs[i] != _oRsrv && _rsrvs[i] != _tRsrv) {
viewVars[3] += dViewNumeraireBalance(_rsrvs[i], _this);
}
}

return viewVars;

}

/// @author james foley http://github.com/realisation
/// @notice calculates the origin amount in an origin trade including the fees
/// @param _oWeight the balance weighting of the origin flavor
/// @param _oBal the new numeraire balance of the origin reserve including the origin amount being swapped
/// @param _oNAmt the origin numeraire amount being swapped
/// @param _grossLiq the numeraire amount across all stablecoin reserves in the contract
/// @return oNAmt_ the origin numeraire amount for the swap with fees applied
function calculateOriginTradeOriginAmount (uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase, uint256
_feeDerivative) external view returns (uint256) {

require(_oBal <= wmul(_oWeight, wmul(_grossLiq, _alpha + WAD)), "origin swap origin halt check");

uint256 oNAmt_;

uint256 _feeThreshold = wmul(_oWeight, wmul(_grossLiq, _beta + WAD));
if (_oBal <= _feeThreshold) {

oNAmt_ = _oNAmt;
```

```

} else if (sub(_oBal, _oNAmt) >= _feeThreshold) {

uint256 _fee = wdiv(
sub(_oBal, _feeThreshold),
wmul(_oWeight, _grossLiq)
);
_fee = wmul(_fee, _feeDerivative);
oNAmt_ = wmul(_oNAmt, WAD - _fee);

} else {

uint256 _fee = wdiv(
sub(_oBal, _feeThreshold),
wmul(_oWeight, _grossLiq)
);

_fee = wmul(_feeDerivative, _fee);

oNAmt_ = add(
sub(_feeThreshold, sub(_oBal, _oNAmt)),
wmul(sub(_oBal, _feeThreshold), WAD - _fee)
);

}

return oNAmt_;

}

/// @author james foley http://github.com/realisation
/// @notice calculates the fees to apply to the target amount in an origin trade
/// @param _tWeight the balance weighting of the target flavor
/// @param _tBal the current balance of the target in the reserve
/// @param _grossLiq the current total balance across all the reserves in the contract
/// @return tNAmt_ the target numeraire amount including any applied fees
function calculateOriginTradeTargetAmount (address _tAdptr, uint256 _tWeight, uint256 _tBal, uint256 _tNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase,
uint256 _feeDerivative) external view returns (uint256 tNAmt_) {

uint256 _feeThreshold = wmul(_tWeight, wmul(_grossLiq, WAD - _beta));

if (sub(_tBal, _tNAmt) >= _feeThreshold) {

tNAmt_ = wmul(_tNAmt, WAD - _feeBase);

} else if (_tBal <= _feeThreshold) {

uint256 _fee = wdiv(
sub(_feeThreshold, sub(_tBal, _tNAmt)),
wmul(_tWeight, _grossLiq)
);
_fee = wmul(_fee, _feeDerivative);
_tNAmt = wmul(_tNAmt, WAD - _fee);
tNAmt_ = wmul(_tNAmt, WAD - _feeBase);

} else {

uint256 _fee = wdiv(
sub(_feeThreshold, sub(_tBal, _tNAmt)),
wmul(_tWeight, _grossLiq)
);

_fee = wmul(_feeDerivative, _fee);
```

```

tNAmt_ = add(
sub(_tBal, _feeThreshold),
wmul(sub(_feeThreshold, sub(_tBal, _tNAmt)), WAD - _fee)
);

tNAmt_ = wmul(tNAmt_, WAD - _feeBase);

}

require(sub(_tBal, tNAmt_) >= wmul(_tWeight, wmul(_grossLiq, WAD - _alpha)), "origin swap target halt check");

return dViewRawAmount(_tAdptr, tNAmt_);

}

function getTargetViewVariables (address _this, address[] calldata _rsrvs, address _oAdptr, address _oRsrv, address _tAdptr, address _tRsrv, uint256 _tAmt) external view returns
(uint256[] memory) {

uint256[] memory viewVars = new uint256[](4);

viewVars[0] = dViewNumeraireAmount(_tAdptr, _tAmt);
viewVars[1] = dViewNumeraireBalance(_tAdptr, _this);
viewVars[3] += viewVars[1];
viewVars[1] -= viewVars[0];

viewVars[2] = dViewNumeraireBalance(_oAdptr, _this);
viewVars[3] += viewVars[2];

for (uint i = 0; i < _rsrvs.length; i++) {
if (_rsrvs[i] != _oRsrv && _rsrvs[i] != _tRsrv) {
viewVars[3] += dViewNumeraireBalance(_rsrvs[i], _this);
}
}

return viewVars;

}

/// @author james foley http://github.com/realisation
/// @notice this function applies fees to the target amount according to how balanced it is relative to its weight
/// @param _tWeight the weighted balance point of the target token
/// @param _tBal the contract's balance of the target
/// @param _tNAmt the numeraire value of the target amount being traded
/// @param _grossLiq the total numeraire value of all liquidity across all the reserves of the contract
/// @return tNAmt_ the target numeraire amount after applying fees
function calculateTargetTradeTargetAmount(uint256 _tWeight, uint256 _tBal, uint256 _tNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase, uint256
_feeDerivative) external view returns (uint256 tNAmt_) {

require(_tBal >= wmul(_tWeight, wmul(_grossLiq, WAD - _alpha)), "target halt check for target trade");

uint256 _feeThreshold = wmul(_tWeight, wmul(_grossLiq, WAD - _beta));
if (_tBal >= _feeThreshold) {

tNAmt_ = wmul(_tNAmt, WAD + _feeBase);

} else if (add(_tBal, _tNAmt) <= _feeThreshold) {

uint256 _fee = wdiv(sub(_feeThreshold, _tBal), wmul(_tWeight, _grossLiq));
_fee = wmul(_fee, _feeDerivative);
_tNAmt = wmul(_tNAmt, WAD + _fee);
tNAmt_ = wmul(_tNAmt, WAD + _feeBase);

```

```
} else {

uint256 _fee = wmul(_feeDerivative, wdiv(
sub(_feeThreshold, _tBal),
wmul(_tWeight, _grossLiq)
));

_tNAmt = add(
sub(add(_tBal, _tNAmt), _feeThreshold),
wmul(sub(_feeThreshold, _tBal), WAD + _fee)
);

tNAmt_ = wmul(_tNAmt, WAD + _feeBase);

}

return tNAmt_;

}

/// @author james foley http://github.com/realisation
/// @notice this function applies fees to the origin amount according to how balanced it is relative to its weight
/// @param _oWeight the weighted balance point of the origin token
/// @param _oBal the contract's balance of the origin
/// @param _oNAmt the numeraire value for the origin amount being traded
/// @param _grossLiq the total numeraire value of all liquidity across all the reserves of the contract
/// @return oNAmt_ the origin numeraire amount after applying fees
function calculateTargetTradeOriginAmount (address _oAdptr, uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase,
uint256 _feeDerivative) external view returns (uint256 oNAmt_) {

uint256 _feeThreshold = wmul(_oWeight, wmul(_grossLiq, WAD + _beta));
if (_oBal + _oNAmt <= _feeThreshold) {

oNAmt_ = _oNAmt;

} else if (_oBal >= _feeThreshold) {

uint256 _fee = wdiv(
sub(add(_oNAmt, _oBal), _feeThreshold),
wmul(_oWeight, _grossLiq)
);
_fee = wmul(_fee, _feeDerivative);
oNAmt_ = wmul(_oNAmt, WAD + _fee);

} else {

uint256 _fee = wmul(_feeDerivative, wdiv(
sub(add(_oBal, _oNAmt), _feeThreshold),
wmul(_oWeight, _grossLiq)
));

oNAmt_ = add(
sub(_feeThreshold, _oBal),
wmul(sub(add(_oBal, _oNAmt), _feeThreshold), WAD + _fee)
);

}

require(add(_oBal, oNAmt_) <= wmul(_oWeight, wmul(_grossLiq, WAD + _alpha)), "origin halt check for target trade");

return dViewRawAmount(_oAdptr, oNAmt_);

}
```

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



}
```

LOW

SWC-128

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

```
z = n % 2 != 0 ? x : RAY;

for (n /= 2; n != 0; n /= 2) {
x = rmul(x, x);
```

LOW

SWC-131

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".

Source file

`src/LoihiViews.sol`

Locations

```
/// @param _grossLiq the numeraire amount across all stablecoin reserves in the contract
/// @return oNAmt_ the origin numeraire amount for the swap with fees applied
function calculateOriginTradeOriginAmount (uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase, uint256
_feeDerivative) external view returns (uint256) {

require(_oBal <= wmul(_oWeight, wmul(_grossLiq, _alpha + WAD)), "origin swap origin halt check");
```

LOW

SWC-131

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".

Source file

`src/LoihiViews.sol`

Locations

```
/// @param _grossLiq the total numeraire value of all liquidity across all the reserves of the contract
/// @return oNAmt_ the origin numeraire amount after applying fees
function calculateTargetTradeOriginAmount (address _oAdptr, uint256 _oWeight, uint256 _oBal, uint256 _oNAmt, uint256 _grossLiq, uint256 _alpha, uint256 _beta, uint256 _feeBase,
uint256 _feeDerivative) external view returns (uint256 oNAmt_) {

uint256 _feeThreshold = wmul(_oWeight, wmul(_grossLiq, WAD + _beta));
```