hexens × astrolab

Apr.23

SECURITY REVIEW REPORT FOR ASTROLAB

CONTENTS

CONTENTS

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

AUDIT LED BY

VAHE KARAPETYAN

Co-founder / CTO | Hexens

Audit Starting Date
03.04.2023

Audit Completion Date
17.04.2023

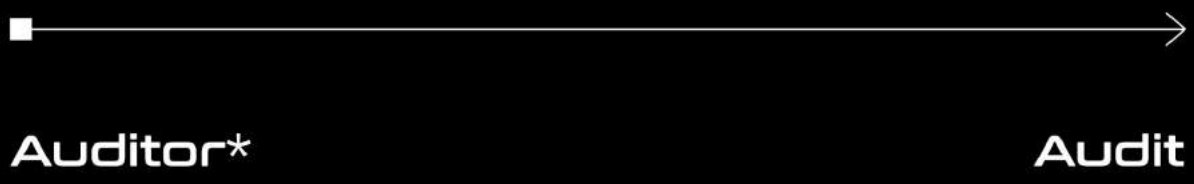
---

hexens × astrolab

METHODOLOGY

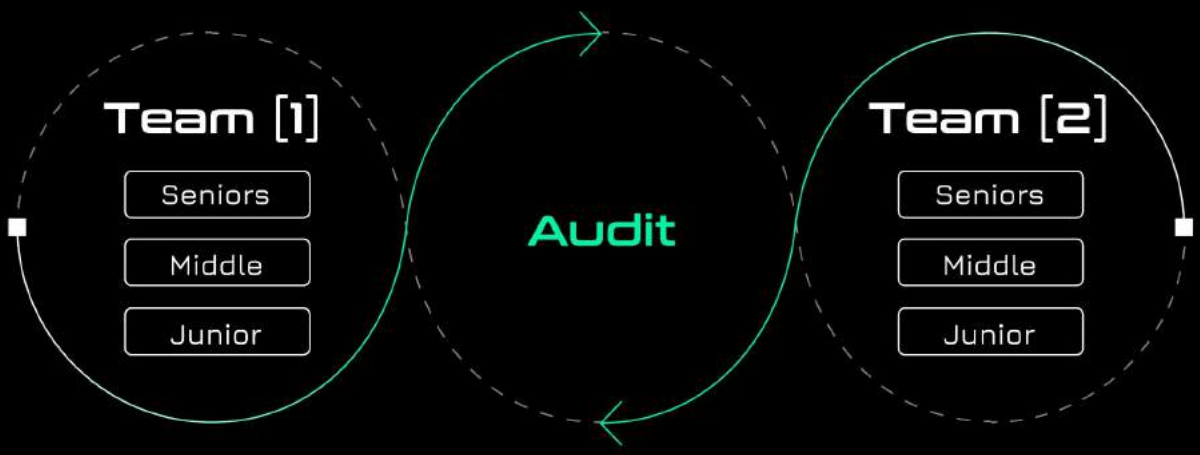
COMMON AUDIT PROCESS

Companies often assign just one engineer to one security assessment with no specified level. Despite the possible impeccable skills of the assigned engineer, it carries risks of the human factor that can affect the product's lifecycle.



HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.



SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

IMPACT	PROBABILITY			
	Rare	Unlikely	Likely	Very Likely
Low / Info	Low / Info	Low / Info	Medium	Medium
Medium	Low / Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of vulnerabilities:

CRITICAL

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

HIGH

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

MEDIUM

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

LOW

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

INFORMATIONAL

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

It's important to consider all types of vulnerabilities, including informational ones, when assessing the security of the project. A comprehensive security audit should consider all types of vulnerabilities to ensure the highest level of security and reliability.

EXECUTIVE SUMMARY

OVERVIEW

This audit covered Astrolab's new cross chain protocol. The protocol is based on special vault system to provide immediate liquidity for cross chain actions and also it uses Stargate.

Our security assessment was a full review of the Astrolab's protocol (except the exact implementations of parent smart contracts like LayerZero App). We have thoroughly reviewed each contract individually, as well as the system as a whole.

During our audit, we have identified 1 critical severity vulnerability in the Crate contract. It would allow to inflate the vault share price and effectively steal assets from other users.

We have also identified 1 high severity vulnerability, various minor vulnerabilities and code optimisations.

Finally, all of our reported issues were fixed by the development team and consequently validated by us.

We can confidently say that the overall security and code quality has increased after completion of our audit.

SCOPE

The analyzed resources are located at:
https://github.com/AstrolabFinance/contracts/commit/ded9dff476f415a5934f2cfcd3d1120734039272

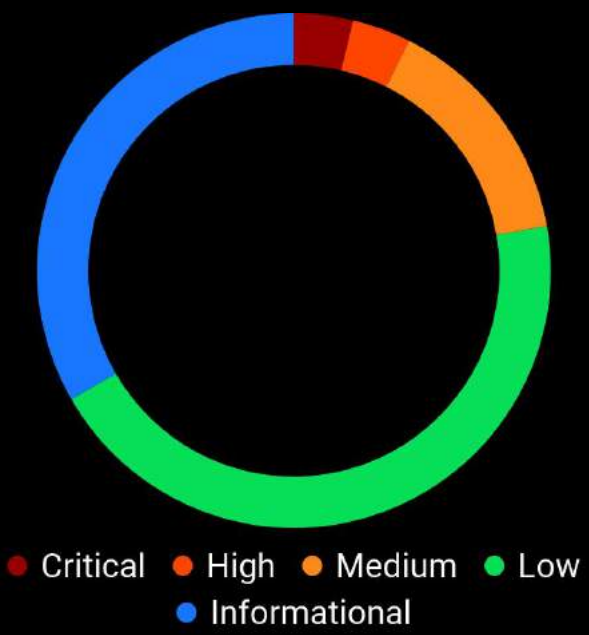
The issues described in this report were fixed in the following commit:
https://github.com/AstrolabFinance/contracts/commit/62335c7fec2870babf0ff6170ccef3603747b42c

SUMMARY

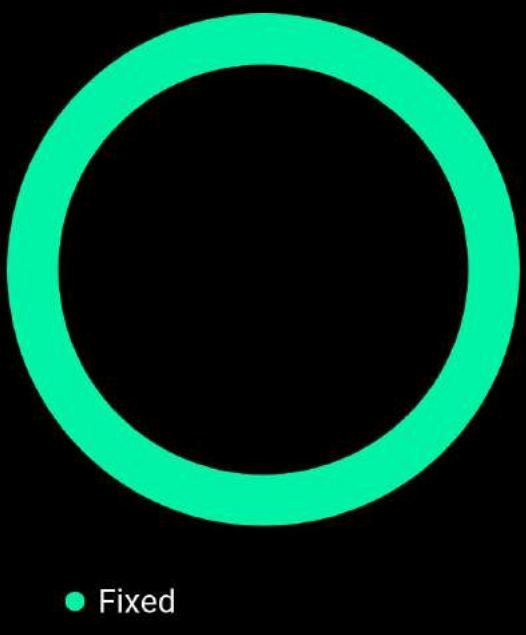
SEVERITY	NUMBER OF FINDINGS
CRITICAL	1
HIGH	1
MEDIUM	4
LOW	12
INFORMATIONAL	9

TOTAL: 21

SEVERITY



STATUS



WEAKNESSES

This section contains the list of discovered weaknesses.

ASTRO-13. WRONG DEBT CALCULATIONS DURING WITHDRAWAL

SEVERITY: Critical

PATH: Crate.sol

REMEDIATION: move decreasing liquididyPool.dept decreasing in try block

STATUS: fixed

DESCRIPTION:

The functions **safeWithdraw**, and **safeRedeem** have inconsistent debt calculation logic.

A bad actor can call **safeWithdraw/safeRedeem** function with such parameters that revert **swapVirtualToAsset** function call (for example, with a past deadline). That function is executed in **try/catch** block, but before the **try/catch the _withdraw()** function decreases **liquidityPool.debt**. Even if **swapToVirtualAsset()** reverts the **liquidityPool.debt** will already be decreased.

As a result, **_withdraw()** function decreases **crate.totalAssets()** twice. It affects all calculations, f.e **increaseLiquidity**, **decreaseLiquidity**, **rebalanceLiquidity**, **shareToAsset**, **assetToShare**, etc.

```
function _withdraw(
    uint256 _amount,
    uint256 _shares,
    uint256 _minAmountOut,
    uint256 _deadline,
    address _receiver,
    address _owner
) internal nonReentrant returns (uint256 recovered) {
    if (_amount == 0 || _shares == 0) revert AmountZero();

    // We spend the allowance if the msg.sender isn't the receiver
    if (msg.sender != _owner) {
        _spendAllowance(_owner, msg.sender, _shares);
    }

    // Check for rounding error since we round down in previewRedeem.
    if (convertToAssets(_shares) == 0)
        revert IncorrectAssetAmount(convertToAssets(_shares));

    // We burn the tokens
    _burn(_owner, _shares);

    // Allows to take a withdraw fee
    _amount = (_amount * (MAX_BPS - withdrawFee)) / MAX_BPS;

    if (liquidityPoolEnabled) {
        // We don't take into account the eventual slippage, since it will
        // be paid to the depositoors
        liquidityPool.debt -= Math.min(_amount, liquidityPool.debt);
        try
            liquidityPool.swap.swapVirtualToAsset(
                _amount,
                _minAmountOut,
                _deadline,
                _receiver
            )
```

```
            returns (uint256 dy) {
                recovered = dy;
            } catch {
                // if the swap fails, we send the funds available
                asset.safeTransfer(_receiver, _amount);
                recovered = _amount;
            }
        } else {
            // If the liquidity pool is not enabled, we send the funds available
            // This allows for the bootstrapping of the pool at start
            asset.safeTransfer(_receiver, _amount);
            recovered = _amount;
        }

        if (_minAmountOut > 0 && recovered < _minAmountOut)
            revert IncorrectAssetAmount(recovered);

        emit Withdraw(msg.sender, _receiver, _owner, _amount, _shares);
        return (recovered);
    }
```

ASTRO-22. NO SLIPPAGE PROTECTION FOR BRIDGEFUNDS

SEVERITY: High

PATH: BridgeConnectorHomeSTG.sol, BridgeConnectorRemoteSTG.sol

REMEDIATION: add a minimal output amount parameter in bridgeFunds

STATUS: fixed

DESCRIPTION:

The function bridgeFunds (in both Home and Remote contracts) is used to send assets from a crate to a remote chain; it uses Stargate to make the swap. Although the minimal output amount parameter of the swap is hardcoded to be zero, thus the swap will be made without slippage protection. Although a stableswap is being made, slippage protection should be in place as it can be “sandwich” attacked or manipulated by unexpected market conditions.

Furthermore, all debt calculations will be misaligned if slippage occurs and thus impact protocol consistency.

```
function bridgeFunds(
    uint256 _amount,
    uint256 _chainId
  ) external payable override onlyCrate {B
    // Loading this in memory for gas savings
    // We send directly to the allocator
    address destination = allocatorsMap[_chainId];
    uint256 dstPoolId = dstPoolIdMap[_chainId];
    if (dstPoolId == 0) {
      revert PoolNotSet(_chainId);
    }

    // Bridging using Stargate
    IStargateRouter(stgEndpoint).swap{ value: msg.value }(
      lzChainIdMap[_chainId], // destination chain id
      srcPoolId, // local pool id (ex: USDC is 1)
      dstPoolIdMap[_chainId], // remote pool id
      payable(tx.origin), // refund address for extra gas
      _amount, // quantity to swap
      0, // the min qty you would accept on the destination
      IStargateRouter.lzTxObj(0, 0, bytes("")), // params for gas forwarding
      abi.encodePacked(destination), // receiver of the tokens
      bytes("") // data for the destination router
    );
    emit BridgeSuccess(_amount, _chainId, msg.value, destination);
  }
```

ASTRO-24. WRONG REBALANCE IMPLEMENTATION

SEVERITY: Medium

PATH: Crate.sol

REMEDIATION: consider rebalancing before each withdrawal to ensure the correctness of the output amount. For not getting reverted, in case of the pool is already rebalanced, consider changing revert from rebalanceLiquidityPool() to f.e if statement implementation

STATUS: fixed

DESCRIPTION:

When a user withdraws their assets or shares using the **withdraw()** or **redeem()** functions, the pool is not being rebalanced. Therefore, if the case of another user also wants to withdraw, they may end up withdrawing the wrong amount of shares since the shares will not be rebalanced at that point. This can result in a lower price for the shares, causing the second user to receive fewer assets than they should. Additionally, the protocol will earn more from this transaction.

```
function _withdraw(
    uint256 _amount,
    uint256 _shares,
    uint256 _minAmountOut,
    uint256 _deadline,
    address _receiver,
    address _owner
) internal nonReentrant returns (uint256 recovered) {
    if (_amount == 0 || _shares == 0) revert AmountZero();

    // We spend the allowance if the msg.sender isn't the receiver
    if (msg.sender != _owner) {
        _spendAllowance(_owner, msg.sender, _shares);
    }

    // Check for rounding error since we round down in previewRedeem.
    if (convertToAssets(_shares) == 0)
        revert IncorrectAssetAmount(convertToAssets(_shares));

    // We burn the tokens
    _burn(_owner, _shares);

    // Allows to take a withdraw fee
    _amount = (_amount * (MAX_BPS - withdrawFee)) / MAX_BPS;

    if (liquidityPoolEnabled) {
        // We don't take into account the eventual slippage, since it will
        // be paid to the depositoors
        liquidityPool.debt -= Math.min(_amount, liquidityPool.debt);
        try
            liquidityPool.swap.swapVirtualToAsset(
                _amount,
                _minAmountOut,
                _deadline,
                _receiver
            )
        returns (uint256 dy) {
            recovered = dy;
        } catch {
            // if the swap fails, we send the funds available
            asset.safeTransfer(_receiver, _amount);
            recovered = _amount;
        }
```

```
    } else {
        // If the liquidity pool is not enabled, we send the funds available
        // This allows for the bootstrapping of the pool at start
        asset.safeTransfer(_receiver, _amount);
        recovered = _amount;
    }

    if (_minAmountOut > 0 && recovered < _minAmountOut)
        revert IncorrectAssetAmount(recovered);

    emit Withdraw(msg.sender, _receiver, _owner, _amount, _shares);
    return (recovered);
}

function rebalanceLiquidityPool()
    public
    whenNotPaused
    returns (uint256 earned)
{
    // Reverts if we the LP is not enabled
    if (!liquidityPoolEnabled) revert LiquidityPoolNotSet();

    // We check if we have enough funds to rebalance
    uint256 toSwap = _getAmountToSwap(
        asset.balanceOf(address(this)),
        liquidityPool
    );

    if (toSwap == 0) revert NoFundsToRebalance();
    uint256 recovered = liquidityPool.swap.swapAssetToVirtual(
        toSwap,
        block.timestamp + 100
    );
    liquidityPool.debt += recovered;
    earned = recovered - Math.min(toSwap, recovered);

    emit LiquidityRebalanced(recovered, earned);
    emit SharePriceUpdated(sharePrice(), block.timestamp);
}
```

ASTRO-8. WRONG EMERGENCY STOP PATTERN USAGE

SEVERITY: Medium

PATH: Crate.sol

REMEDIATION: use whenNotPaused modifier for safeWithdraw(), redeem(), and safeRedeem() functions

STATUS: fixed

DESCRIPTION:

The contract **Crate.sol** provides an **Emergency Stop pattern**, but some functions don't have any checks to halt their usage, so users can withdraw or redeem after the owner has paused the contract.

```
function safeWithdraw(
    uint256 _amount,
    uint256 _minAmount,
    uint256 _deadline,
    address _receiver,
    address _owner
  ) external returns (uint256 shares) {
    // This represents the amount of crTokens that we're about to burn
    shares = convertToShares(_amount);
    _withdraw(_amount, shares, _minAmount, _deadline, _receiver, _owner);
  }
```

```
function redeem(
    uint256 _shares,
    address _receiver,
    address _owner
  ) external returns (uint256 assets) {
    return (
      _withdraw(
        convertToAssets(_shares),
        _shares,
        0,
        block.timestamp,
        _receiver,
        _owner
      )
    );
  }
```

```
function safeRedeem(
    uint256 _shares,
    uint256 _minAmountOut, // Min_amount
    uint256 _deadline,
    address _receiver,
    address _owner
  ) external returns (uint256 assets) {
    return (
      _withdraw(
        convertToAssets(_shares), // _amount
        _shares, // _shares
        _minAmountOut,
        _deadline,
        _receiver, // _receiver
        _owner // _owner
      )
    );
  }
```

ASTRO-19. MISSING FEE LIMITS CHECK

SEVERITY: Medium

PATH: Crate.sol

REMEDIATION: add checks for fees in the contract constructor to ensure that they are not bigger than MAX_PERF_FEE, MAX_MGMT_FEE and MAX_WITHDRAW_FEE values

STATUS: fixed

DESCRIPTION:

Unlike in the setFees function, the Crate contract constructor is setting the withdraw, performance and management fees, which are not being checked to be smaller than some amount (e.g. 10%); checks should exist ensuring that the fees are not exceeding some appropriate amount.

This way, the contract creator can set big fee values, even those that will surpass 100% and drain the assets via a fee mechanism.

```
constructor(
    address _asset, // The asset we are using
    string memory _name, // The name of the token
    string memory _symbol, // The symbol of the token
    uint256 _performanceFee, // 100% = 10000
    uint256 _managementFee, // 100% = 10000
    uint256 _withdrawFee // 100% = 10000
) ERC20(_name, _symbol) {
    asset = IERC20(_asset);
    performanceFee = _performanceFee;
    managementFee = _managementFee;
    withdrawFee = _withdrawFee;
    tokenDecimals = IERC20Metadata(_asset).decimals();
    checkpoint = Checkpoint(block.timestamp, 10 ** tokenDecimals);
    _pause(); // We start paused
}
```

ASTRO-20. INCORRECT WITHDRAW PREVIEW CALCULATIONS

SEVERITY: Medium

PATH: Crate.sol

REMEDIATION: change the preview formula to convertToShares((_assets * MAX_BPS)/(MAX_BPS-withdrawfee))

STATUS: fixed

DESCRIPTION:

In the contract Crate, the withdraw preview function miscalculates how many shares are needed to withdraw a fixed amount of assets.

The _withdraw function calculates the output amount:

_amount = (_amount * (MAX_BPS - withdrawFee)) / MAX_BPS;

while previewFunction uses the following formula:

(convertToShares(_assets) * (MAX_BPS + withdrawFee)) / MAX_BPS

Whilst the correct way to calculate the result will be to use formula:

convertToShares((_assets * MAX_BPS)/(MAX_BPS-withdrawfee)

As the majority of users will be using the frontend to preview and make the withdrawals, this will eventually make them burn incorrect amounts of shares.

```
function previewWithdraw(uint256 _assets) public view returns (uint256) {
    return (convertToShares(_assets) * (MAX_BPS + withdrawFee)) / MAX_BPS;
}
```

ASTRO-14. MISSING EVENT ON DECREASING LIQUIDITY

SEVERITY: Low

PATH: Crate.sol

REMEDIATION: add LiquidityChanged event emission

STATUS: fixed

DESCRIPTION:

The function **decreaseLiquidity** does not emit the event **LiquidityChanged**, in case there is such an event, in contrary to the function **increaseLiquidity** which does emit the event.

```
function decreaseLiquidity(uint256 _liquidityRemoved) external onlyOwner {
    // Rebalance first the pool to avoid any negative slippage
    uint256 lpBal = liquidityPool.swap.getVirtualLpBalance();
    uint256 assetBalBefore = asset.balanceOf(address(this));
    uint256 liquidityBefore = liquidityPool.liquidity;

    // we remove liquidity
    liquidityPool.liquidity -= _liquidityRemoved;
    // We specify the amount of LP that corresponds to the amount of liquidity removed
    liquidityPool.swap.removeLiquidity(
        (lpBal * _liquidityRemoved) / liquidityBefore,
        block.timestamp
    );
    // We update the book
    liquidityPool.debt -= (asset.balanceOf(address(this)) - assetBalBefore);
}
```

ASTRO-15. POSSIBLE FREEZE OF NATIVE TOKENS

SEVERITY: Low

PATH: Allocator.sol, BridgeConnectorHomeSTG.sol

REMEDIATION: consider removing the receive() function

STATUS: fixed

DESCRIPTION:

The contract Allocator and BridgeConnectorHomeSTG, despite having payable functions that accept ether (or other native tokens), also have a payable fallback function.

Nevertheless, the contracts are not designed to accept native tokens other than in payable functions.

In the case of an Allocator contract, there is no function implemented to sweep the wrongfully sent native tokens. Thus the tokens might eventually get “frozen” in the contract. The BridgeConnectorHomeSTG contract has the sweeping function, but the logic is overall redundant.

```
receive() external payable {}
```

ASTRO-5. REDUNDANT IMPORTS

SEVERITY: Low

PATH: Swap.sol, Crate.sol

REMEDIATION: remove the redundant imports in favour of a smaller contract size and clean code considerations

STATUS: fixed

DESCRIPTION:

1. Swap.sol: import "@openzeppelin/contracts/security/Pausable.sol" L7
2. Crate.sol: import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol" L15
3. Allocator.sol: import "./interfaces/ICrate.sol" L15

Contract **Swap.sol** imported the **Pausable.sol** but has never used it.

Contract **Crate.sol** imported the **IERC20Metadata.sol** but has never used it.

Contract **Allocator.sol** imported the **ICrate.sol** but has never used it.

```
import "@openzeppelin/contracts/security/Pausable.sol";
```

```
import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
```

```
import "./interfaces/ICrate.sol";
```

ASTRO-1. MISSING ARRAY PARAMETERS LENGTH CHECK

SEVERITY: Low

PATH: Crate.sol, Allocator.sol

REMEDIATION: see description

STATUS: fixed

DESCRIPTION:

In the following locations, there are at least 2 arrays as function parameters that require the same length but are missing these length checks:

1. contracts/Crate.sol:dispatchAssets (L479-500)
2. contracts/Crate.sol:estimateDispatchCost (L1037-1049)
3. contracts/Allocator.sol:dispatchAssets(L182-203)

```
contract Crate {

  [...]

  function dispatchAssets(
      uint256[] calldata _amounts,
      uint256[] calldata _chainIds,
      uint256[] calldata _msgValues
  ) external payable onlyOwner {
      for (uint256 i = 0; i < _amounts.length; i++) {
          ChainData memory data = chainData[_chainIds[i]];
          // checks
          if (data.maxDeposit == 0) revert ChainError(); // Chain not active
          if (data.maxDeposit <= data.debt + _amounts[i])
              revert AmountTooHigh(data.maxDeposit); // No more funds can be sent to this chain

          chainData[_chainIds[i]].debt += _amounts[i];
          totalRemoteAssets += _amounts[i];
          asset.safeTransfer(data.bridge, _amounts[i]);
          if (block.chainid != _chainIds[i]) {
              IBridgeConnectorHome(data.bridge).bridgeFunds{
                  value: _msgValues[i]
              }(_amounts[i], _chainIds[i]);
          }
      }
  }

  [...]
```

```
  function estimateDispatchCost(
      uint256[] calldata _chainIds,
      uint256[] calldata _amounts
  ) external view returns (uint256[] memory) {
      uint256 length = _chainIds.length;
      uint256[] memory nativeCost = new uint256[](length);
      for (uint256 i; i < length; i++) {
          if (_chainIds[i] == block.chainid) continue;
          nativeCost[i] = IBridgeConnectorHome(chainData[_chainIds[i]].bridge)
              .estimateBridgeCost(_chainIds[i], _amounts[i]);
      }
      return (nativeCost);
  }

  [...]
}
Contract Allocator {
[...]
  function dispatchAssets(
      uint256[] calldata _amounts,  // @audit array lenghts
      address[] calldata _strategies
  ) external onlyOwner {
      for (uint256 i; i < _amounts.length; i++) {
          isWhitelisted(_strategies[i]);

          uint256 debt = strategiesData[_strategies[i]].debt;

          if (strategiesData[_strategies[i]].maxDeposit <= debt + _amounts[i])
              revert MaxDepositReached(_strategies[i]);

          strategiesData[_strategies[i]].debt += _amounts[i];
          totalStrategysDebt += _amounts[i];

          IERC20(asset).safeTransfer(_strategies[i], _amounts[i]);

          emit StrategyUpdate(_strategies[i], debt += _amounts[i]);
      }

      emit ChainDebtUpdate(totalChainDebt());
  }
```

```
[...]

}
```

We would recommend checking the length of the input parameters so that the error is caught to improve user experience.

For example:

```
require(array1.length == array2.length, "Arrays length mismatch");
```

ASTRO-23. CODE STYLE PROBLEMS

SEVERITY: Low

PATH: Crate.sol, Swap.sol

REMEDIATION: consider tightly packing variable declarations and removing the lines with the return keywords

STATUS: fixed

DESCRIPTION:

In the Crate contract, the ordering of the variable declaration can be tightly packed to optimize storage usage and save gas.

In the Crate and Swap contracts there is redundant **return** keywords usage as the variables are declared with the same syntax in the function's **return tuple**.

```
uint8 private tokenDecimals; // The decimals of the token
uint256 public totalRemoteAssets; // Amount of assets on other chains (or farmed on local chain)
uint256 public performanceFee; // 100% = 10000
uint256 public managementFee; // 100% = 10000
uint256 public withdrawFee; // 100% = 10000
uint256 public anticipatedProfits; // The yield trickling down
uint256 public lastUpdate; // Last time the unrealized gain was updated
Checkpoint public checkpoint; // Used to compute fees
ElasticLiquidityPool public liquidityPool; // The pool used to process withdraws
bool public liquidityPoolEnabled; // If the pool is enabled
IERC20 public asset; // The asset we are using
```

```
function mint(
    uint256 _shares,
    address _receiver
  ) external nonReentrant returns (uint256 assets) {
    assets = convertToAssets(_shares);

    // Requires
    if (assets == 0 || _shares == 0) revert AmountZero();
    if (assets > maxDeposit(_receiver))
       revert AmountTooHigh(maxDeposit(_receiver));
    if (_receiver == address(this)) revert CrateCantBeReceiver();

    // Moving value
    asset.safeTransferFrom(msg.sender, address(this), assets);
    _mint(_receiver, _shares);
    emit Deposit(msg.sender, _receiver, assets, _shares);
    return (assets);
  }
```

```
function _withdraw(
      uint256 _amount,
      uint256 _shares,
      uint256 _minAmountOut,
      uint256 _deadline,
      address _receiver,
      address _owner
   ) internal nonReentrant returns (uint256 recovered) {
      if (_amount == 0 || _shares == 0) revert AmountZero();

      // We spend the allowance if the msg.sender isn't the receiver
      if (msg.sender != _owner) {
         _spendAllowance(_owner, msg.sender, _shares);
      }

      // Check for rounding error since we round down in previewRedeem.
      if (convertToAssets(_shares) == 0)
         revert IncorrectAssetAmount(convertToAssets(_shares));

      // We burn the tokens
      _burn(_owner, _shares);

      // Allows to take a withdraw fee
      _amount = (_amount * (MAX_BPS - withdrawFee)) / MAX_BPS;

      if (liquidityPoolEnabled) {
         // We don't take into account the eventual slippage, since it will
         // be paid to the depositoors
         liquidityPool.debt -= Math.min(_amount, liquidityPool.debt);
         try
            liquidityPool.swap.swapVirtualToAsset(
               _amount,
               _minAmountOut,
               _deadline,
               _receiver
            )
```

```
        returns (uint256 dy) {
            recovered = dy;
        } catch {
            // if the swap fails, we send the funds available
            asset.safeTransfer(_receiver, _amount);
            recovered = _amount;
        }
    } else {
        // If the liquidity pool is not enabled, we send the funds available
        // This allows for the bootstrapping of the pool at start
        asset.safeTransfer(_receiver, _amount);
        recovered = _amount;
    }

    if (_minAmountOut > 0 && recovered < _minAmountOut)
        revert IncorrectAssetAmount(recovered);

    emit Withdraw(msg.sender, _receiver, _owner, _amount, _shares);
    return (recovered);
}
```

```
function swapVirtualToAsset(
    uint256 _dx,
    uint256 _minDy,
    uint256 _deadline,
    address _receiver
)
    external
    deadlineCheck(_deadline)
    onlyCrate
    returns (uint256 dy)
{
    // If we are swapping 0, we return 0
    if (_dx == 0) {
        return 0;
    }

    dy = swapStorage._swap(
        VIRTUAL_ASSET_INDEX,
        REAL_ASSET_INDEX,
        _dx,
        _minDy
    );

    LENDING_POOL.withdraw(
        address(UNDERLYING_TOKENS[REAL_ASSET_INDEX]),
        dy,
        _receiver
    );

    // And we withdraw and send them to the recipient
    return dy;
}
```

ASTRO-4. REDUNDANT USAGE OF SAFEMATH

SEVERITY: Low

PATH: Swap.sol

REMEDIATION: remove the import of OpenZeppelin's SafeMath library and replace all the arithmetic operations with their Solidity equivalent

STATUS: fixed

DESCRIPTION:

The contract uses Solidity **0.8.17**, which already checks arithmetic operations for over- and underflow. This is suboptimal and wastes gas on each function call containing these operations.

```
import "@openzeppelin/contracts/utils/math/SafeMath.sol";
[...]
contract Swap is Ownable {
  using SafeMath for uint256;
  [...]
}
```

ASTRO-6. LACK OF PARAMETER DESCRIPTION

SEVERITY: Informational

PATH: SwapUtils.sol

REMEDIATION: add a description for _mintToMint

STATUS: fixed

DESCRIPTION:

This function **storage** has descriptions of two of its three parameters, the purpose of the third parameter **_mintToMint** is missing in the natspec.

```
/**
    * @notice Add liquidity to the pool
    * @param self Swap struct to read from and write to
    * @param amounts the amounts of each token to add, in their native precision
    * should mint, otherwise revert. Handy for front-running mitigation
    * allowed addresses. If the pool is not in the guarded launch phase, this parameter will be ignored.
    * @return amount of LP token user received
    */
  function _addLiquidity(
      Swap storage self,
      uint256[] memory amounts,
      uint256 _mintToMint
  ) internal returns (uint256) {
```

ASTRO-9. REDUNDANT PAYABLE DECLARATION

SEVERITY: Informational

PATH: Crate.sol

REMEDIATION: remove payable modifier

STATUS: fixed

DESCRIPTION:

This function **updateChainDebt** is declared as payable, although it does not use msg.value and does not call any other payable function.

```
function updateChainDebt(
    uint256 _chainId,
    uint256 _newDebt
  ) external payable onlyBridgeConnector {
    uint256 oldDebt = chainData[_chainId].debt;

    chainData[_chainId].debt = _newDebt;
    uint256 debtDiff = _newDebt - Math.min(_newDebt, oldDebt);
    if (debtDiff > 0) {
      // We update the anticipated profits
      anticipatedProfits = debtDiff + unrealizedGains();
      lastUpdate = block.timestamp;
    }

    totalRemoteAssets = totalRemoteAssets + _newDebt - oldDebt;

    emit ChainDebtUpdated(_newDebt, oldDebt, _chainId);
    emit SharePriceUpdated(sharePrice(), block.timestamp);
  }
```

ASTRO-21. GAS OPTIMISATION

SEVERITY: Informational

PATH: BridgeConnectorHomeSTG.sol, BridgeConnectorRemoteSTG.sol

REMEDIATION: consider adding “immutable” keyword to the srcPoolId variable declaration

STATUS: partially fixed

DESCRIPTION:

In the contract, **BridgeConnectorHomeSTG**, the storage variable **srcPoolId** is being set only in the constructor without the possibility of updating it in future.

Thus using the immutable modifier will help optimize the gas usage of the contract, and it will hardcode the **srcPoolId** in the constructor.

In the contract **BridgeConnectorHomeSTG** the variables **dstPoolId**, **srcPoolId**, **brigeGasAmount**, **homeBridge** and **layerZeroEndpoint** can also be flagged “immutable” and favour gas optimization.

```
constructor(
    address _crate,
    address _asset,
    address _stgEndpoint,
    address _lzEndpoint,
    uint256 _srcPoolId,
    uint16 homeLzChainId
) NonblockingLzApp(_lzEndpoint) {
    crate = _crate;
    asset = _asset;
    stgEndpoint = _stgEndpoint;
    srcPoolId = _srcPoolId;
    convertLzChainId[homeLzChainId] = block.chainid;
    _giveAllowances(_stgEndpoint, _asset);
}
```

```
uint256 public dstPoolId;
uint256 public srcPoolId;
uint256 public brigeGasAmount;
uint256 public updateGasAmount;

address public allocator;
address public homeBridge;

IStargateRouter public immutable stgRouter;
ILayerZeroEndpoint public layerZeroEndpoint;
```

ASTRO-17. CLEAN CODE

SEVERITY: Informational

PATH: Allocator.sol

REMEDIATION: consider using storage reference variable

STATUS: fixed

DESCRIPTION:

In the dispatchAssets function, the **strategiesData[_strategies[i]]** storage mapping variable is used multiple times. For the clean code purposes it is recommended to use storage reference variable in such cases, e.g:

```
Strategy storage strategyData = strategiesData[_strategies[i]];
```

```
function dispatchAssets(
    uint256[] calldata _amounts,
    address[] calldata _strategies
) external onlyOwner {
    for (uint256 i; i < _amounts.length; i++) {
        isWhitelisted(_strategies[i]);

        uint256 debt = strategiesData[_strategies[i]].debt;

        if (strategiesData[_strategies[i]].maxDeposit <= debt + _amounts[i])
            revert MaxDepositReached(_strategies[i]);

        strategiesData[_strategies[i]].debt += _amounts[i];
        totalStrategysDebt += _amounts[i];

        IERC20(asset).safeTransfer(_strategies[i], _amounts[i]);

        emit StrategyUpdate(_strategies[i], debt += _amounts[i]);
    }

    emit ChainDebtUpdate(totalChainDebt());
}
```

ASTRO-18. REDUNDANT INHERITANCE

SEVERITY: Informational

PATH: Crate.sol

REMEDIATION: consider removing the ERC20 inheritance leaving only ERC20Snapshot, as well as removing Ownable inheritance

STATUS: fixed

DESCRIPTION:

The contract Crate inherits both from ERC20 and ERC20Snapshot, although the contract ERC20Snapshot is already derived from ERC20; thus, the ERC20 inheritance is redundant.

contract Crate is Pausable, ReentrancyGuard, Ownable, ERC20Snapshot

```
contract BridgeConnectorHomeSTG is
  Ownable
  NonblockingLzApp,
  IStargateReceiver,
  IBridgeConnectorHome
```

ASTRO-16. INCORRECT EVENT EMISSION PARAMETER

SEVERITY: Informational

PATH: Allocator.sol

REMEDIATION: change the event emission parameter to log crateChainId

STATUS: fixed

DESCRIPTION:

In the Allocator contract's **bridgeBackFunds** function the **BridgeSuccessevent** should rather emit crateChainId in case the funds are bridged back, furthermore as the block.chainid is deterministic for every deployed contract there is no actual need to emit it in the event.

```
function bridgeBackFunds(uint256 _amount) external payable onlyOwner {
    address bc = bridgeConnector;

    //We send assets to the bridge/crate
    IERC20(asset).safeTransfer(bc, _amount);

    if (block.chainid != crateChainId) {
        IBridgeConnectorRemote(bc).bridgeFunds{ value: msg.value }(_amount);
    } else {
        // We update the crate directly if we're on the same chain
        _updateCrate();
    }

    emit BridgeSuccess(_amount, block.chainid);
}
```

ASTRO-12. GAS OPTIMISATION

SEVERITY: Informational

PATH: Allocator.sol

REMEDIATION: consider using calldata for arrays and strings; this will optimize the gas usage

STATUS: fixed

DESCRIPTION:

In the Allocator contract, the function addNewStrategy() takes **_strategyName** as a parameter. The string is declared as memory, which uses more gas for additional memory operations.

```
contract Allocator is Initializable, OwnableUpgradeable {

  [...]

  function addNewStrategy(
     address _entryPoint,
     uint256 _maxDeposit,
     string memory _strategyName
  ) external onlyOwner {
     [...]
  }

  [...]

}
```

ASTRO-11. MISLEADING ERROR DECLARATION

SEVERITY: Informational

PATH: bridgeConnectorRemoteSTG.sol

REMEDIATION: reconsider the error declaration

STATUS: fixed

DESCRIPTION:

The contract is declaring an **error ZeroAddress()** and using it in the **function setAllocator()**(L134-138), but that function doesn't do a zero address check. Instead, it checks whether the **allocator** is set; thereby, it needs a clearer error declaration, e.g., AllocatorSet.

```
contract BridgeConnectorRemoteSTG {

[...]

  error ZeroAddress();

[...]

  function setAllocator(address _allocator) external onlyOwner {
    // We can't change it after it's set
    if (allocator != address(0)) revert ZeroAddress();
    allocator = _allocator;
  }
}
```

ASTRO-2. MESSAGE VALUE CHECK IS MISSING

SEVERITY: Informational

PATH: Crate.sol

REMEDIATION: add a check that all of the_msgValues total to msg.value

STATUS: fixed

DESCRIPTION:

The function dispatchAssets is using a payable modifier meaning that the owner can send **value** to the contract. Additionally, this function sends value to the bridge. But the contract doesn't check if the owner sent enough **value** by calling this function, so the contract might be able to send value from it's own balance, which can be done accidentally. Furthermore the owner might send an exceeding amount of msg.value to the contract.

```
function dispatchAssets(
    uint256[] calldata _amounts,
    uint256[] calldata _chainIds,
    uint256[] calldata _msgValues
  ) external payable onlyOwner {
    for (uint256 i = 0; i < _amounts.length; i++) {
      ChainData memory data = chainData[_chainIds[i]];
      // checks
      if (data.maxDeposit == 0) revert ChainError(); // Chain not active
      if (data.maxDeposit <= data.debt + _amounts[i])
        revert AmountTooHigh(data.maxDeposit); // No more funds can be sent to this chain

      chainData[_chainIds[i]].debt += _amounts[i];
      totalRemoteAssets += _amounts[i];
      asset.safeTransfer(data.bridge, _amounts[i]);
      if (block.chainid != _chainIds[i]) {
        IBridgeConnectorHome(data.bridge).bridgeFunds{
          value: _msgValues[i]
        }(_amounts[i], _chainIds[i]);
      }
    }
  }
```


hexens