METATRUST

Security Assessment for

Hope-lend-code

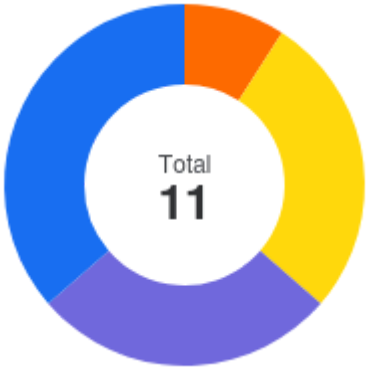
July 25, 2023

Executive Summary

Overview	
Project Name	Hope-lend-code
Codebase URL	https://github.com/Light-Ecosystem/lend-core/tree/audit
Scan Engine	Security Analyzer
Scan Time	2023/07/25 15:06:58
Commit Id	4738b8dc0fb2fd9e628cdee79d4fc1132744f8c6

Total	
Critical Issues	0
High risk Issues	1
Medium risk Issues	3
Low risk Issues	3
Informational Issues	4

Critical Issues	The issue can cause large economic losses, large-scale data disorder, loss of control of authority management, failure of key functions, or indirectly affect the correct operation of other smart contracts interacting with it.
High Risk Issues	The issue puts a large number of users' sensitive information at risk or is reasonably likely to lead to catastrophic impacts on clients' reputations or serious financial implications for clients and users.
Medium Risk Issues	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk Issues	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational Issue	The issue does not pose an immediate risk but is relevant to security best practices or Defence in Depth.



Critical Issues	0%	0
High risk Issues	9%	1
Medium risk Issues	27%	3
Low risk Issues	27%	3
Informational Issues	36%	4

Summary of Findings

MetaScan security assessment was performed on **July 25, 2023 15:06:58** on project **Hope-lend-code** with the repository **Hope_lend_code_Jun_27_v2** on branch **-**. The assessment was carried out by scanning the project's codebase using the scan engine **Security Analyzer**. There are in total **11** vulnerabilities / security risks discovered during the scanning session, among which **0** critical vulnerabilities, **1** high risk vulnerabilities, **3** medium risk vulnerabilities, **3** low risk vulnerabilities, **4** informational issues.

ID	Description	Severity	Alleviation
MSA-001	Wrong variable debt token balance	High risk	Fixed
MSA-002	Missing invoking the `updateAllocation` function	Medium risk	Fixed
MSA-003	Lack of resetting `newRate` if `isKilled` is true	Medium risk	Fixed
MSA-004	The `kick` function may make users loss their reward	Medium risk	Acknowledged
MSA-005	Lack of checking the `failoverActive`	Low risk	Acknowledged
MSA-006	The event mismatches the implementation	Low risk	Fixed
MSA-007	Gas optimization	Low risk	Acknowledged
MSA-008	Unused Parameter	Informational	Fixed
MSA-009	Missing emit event	Informational	Acknowledged
MSA-010	The potential inconsistent between `lendingGauge` and `allLendingGauges`	Informational	Acknowledged
MSA-011	The return value of `_getAllocationByUtilizationRate` function might be misleading under the edge case	Informational	Acknowledged

Findings

Critical (0)

No Critical vulnerabilities found here

High risk (1)

1. Wrong variable debt token balance

High risk Security Analyzer

In `kick` function, the `_variableDebtTokenBalance` is assigned as below, which is wrong:

```
function kick(address _addr) external {
  ...
  uint256 _hTokenBalance = IHTokenRewards(hToken).lpBalanceOf(_addr);
  uint256 _variableDebtTokenBalance = IHTokenRewards(hToken).lpBalanceOf(_addr);
  ...
}
```

The `IHTokenRewards(hToken).lpBalanceOf(_addr)` gets the `hToken` balance for the `addr` instead of the variable debt token balance.

File(s) Affected

contracts/protocol/gauge/LendingGauge.sol #242-257

```
242   function kick(address _addr) external {
243     uint256 _hTokenLast = IHTokenRewards(hToken).integrateCheckpointOf(_addr);
244     uint256 _variableDebtTokenLast = IVariableDebtTokenRewards(variableDebtToken).integrateCheckpointO
245     uint256 _tVe = votingEscrow.userPointHistoryTs(_addr, votingEscrow.userPointEpoch(_addr));
246     uint256 _hTokenBalance = IHTokenRewards(hToken).lpBalanceOf(_addr);
247     uint256 _variableDebtTokenBalance = IHTokenRewards(hToken).lpBalanceOf(_addr);
248
249     require(votingEscrow.balanceOfAtTime(_addr, block.timestamp) == 0 || _tVe > _hTokenLast || _tVe > _
250     require(
251       IHTokenRewards(hToken).workingBalances(_addr) > (_hTokenBalance * 40) / 100 ||
252         IVariableDebtTokenRewards(variableDebtToken).workingBalances(_addr) > (_variableDebtTokenBalanc
253       'GP001'
254     );
255
256     _userCheckpoint(_addr);
257   }
```

Recommendation

Consider correcting the statement used to query the variable debt token balance.

Alleviation Fixed

The development team resolved this issue in commit 82b443d7b507b4a6f1024c2df5302ab9556d7991

Medium risk (3)

1. Missing invoking the `updateAllocation` function

Medium risk Security Analyzer

Function **updateAllocation** needs to be invoked when there is any update on the **stableDebtTokenTotalSupply**, **variableDebtTokenTotalSupply**, and **availableLiquidity**:

```
function updateAllocation() external override onlyPool returns (bool) {
  uint256 stableDebtTokenTotalSupply = IERC20(stableDebtToken).totalSupply();
  uint256 variableDebtTokenTotalSupply = IERC20(variableDebtToken).totalSupply();
  uint256 totalDebt = stableDebtTokenTotalSupply + variableDebtTokenTotalSupply;
  if (totalDebt == 0) {
    borrowAllocation = 0;
    return true;
  }
  uint256 availableLiquidity = IERC20(underlyingAsset).balanceOf(hToken);
  uint256 availableLiquidityPlusDebt = availableLiquidity + totalDebt;
  borrowAllocation = _getAllocationByUtilizationRate(totalDebt.rayDiv(availableLiquidityPlusDebt));
  return true;
}
```

The **executeMintToTreasury** function of the **PoolLogic** contract invokes the **mintToTreasury** function of the HToken:

```
//PoolLogic.sol
function executeMintToTreasury(
  mapping(address => DataTypes.ReserveData) storage reservesData,
  address[] calldata assets
) external {
      ...
      IHToken(reserve.hTokenAddress).mintToTreasury(amountToMint, normalizedIncome);
      ...
    }
  }
}

//HToken.sol
function mintToTreasury(uint256 amount, uint256 index) external virtual override onlyPool {
  ...
  if (feeToVault != address(0) && feeToVaultPercent != 0) {
    uint256 amountToVault = amount.percentMul(feeToVaultPercent);
    IERC20(_underlyingAsset).safeTransfer(feeToVault, amountToVault);
    _mintScaled(address(POOL), _treasury, amount - amountToVault, index);
  } else {
    _mintScaled(address(POOL), _treasury, amount, index);
  }
}
```

There are some underlying assets that are transferred to the **feeToVault** if **feeToVault** and **feeToVaultPercent** are non-zero, which implies that the **updateAllocation** function needs to be invoked at this point.

File(s) Affected

contracts/protocol/gauge/LendingGauge.sol #119-131

```
119   function updateAllocation() external override onlyPool returns (bool) {
120     uint256 stableDebtTokenTotalSupply = IERC20(stableDebtToken).totalSupply();
121     uint256 variableDebtTokenTotalSupply = IERC20(variableDebtToken).totalSupply();
122     uint256 totalDebt = stableDebtTokenTotalSupply + variableDebtTokenTotalSupply;
123     if (totalDebt == 0) {
124       borrowAllocation = 0;
125       return true;
126     }
127     uint256 availableLiquidity = IERC20(underlyingAsset).balanceOf(hToken);
128     uint256 availableLiquidityPlusDebt = availableLiquidity + totalDebt;
129     borrowAllocation = _getAllocationByUtilizationRate(totalDebt.rayDiv(availableLiquidityPlusDebt));
130     return true;
131   }
```

contracts/protocol/tokenization/HToken.sol #105-109

```
105     if (feeToVault != address(0) && feeToVaultPercent != 0) {
106       uint256 amountToVault = amount.percentMul(feeToVaultPercent);
107       IERC20(_underlyingAsset).safeTransfer(feeToVault, amountToVault);
108       _mintScaled(address(POOL), _treasury, amount - amountToVault, index);
109     } else {
```

Recommendation

1. Consider invoking the `updateAllocation` function for every function that updates `stableDebtTokenTotalSupply`, `variableDebtTokenTotalSupply`, and `availableLiquidity`.
2. Or minting HToken to the `feeToVault` to avoid transferring the underlying asset and then avoid invoking the `updateAllocation` from the `executeMintToTreasury` function.

Alleviation Fixed

The development team resolved this issue by invoking the `updateAllocation` function in commit 3fd921a60f5d5d267bb274c8c998c51d45436177

2. Lack of resetting `newRate` if `isKilled` is true

Medium risk Security Analyzer

The `isKilled` is a switch to turn on or turn off the incentive for `LT` token. If the `isKilled` is true, the `rate` will be set to 0 to not distribute the `LT` token reward:

```
//LendingGauge.sol
function hvCheckpoint(address _addr) public override {
  ...
  if (isKilled) {
    // Stop distributing inflation as soon as killed
    _st.rate = 0;
  }
  ...
  if (IHTokenRewards(hToken).totalSupply() != 0) {
    IHTokenRewards(hToken).checkpoint(_addr, _calRelativeWeightByAllocation(hToken), _st);
  }
  if (IVariableDebtTokenRewards(variableDebtToken).totalSupply() != 0)
    IVariableDebtTokenRewards(variableDebtToken).checkpoint(_addr, _calRelativeWeightByAllocation(variableDebtToken), _st
  }
  ...
}

//AbsGauge.sol
function _checkpoint(
  address _addr,
```

```
    uint256 _allocation,
    DataTypes.CheckPointParameters memory _st
  ) internal {
    ...
        if (_workingSupply > 0) {
          if (_st.prevFutureEpoch >= _prevWeekTime && _st.prevFutureEpoch < _weekTime) {
            _integrateInvSupply += (_st.rate * _w * _allocation * (_st.prevFutureEpoch - _prevWeekTime)) / _workingSupply /
            _st.rate = _st.newRate;
            _integrateInvSupply += (_st.rate * _w * _allocation * (_weekTime - _st.prevFutureEpoch)) / _workingSupply / Wad
          } else {
            _integrateInvSupply += (_st.rate * _w * _allocation * _dt) / _workingSupply / WadRayMath.RAY;
          }
        }
        ...
  }
```

From the **`hvCheckpoint`** function of **`LendingGauge`** and the **`_checkpoint`** function of **`AbsGauge`**, we found that the **`_integrateInvSupply`** will not increase if **`_st.prevFutureEpoch`** is greater than **`_weekTime`**.

But, the **`newRate`** will be used to increase the **`_integrateInvSupply`** if **`_st.prevFutureEpoch`** is greater than **`_prevWeekTime`** and **`_st.prevFutureEpoch`** is less than **`_weekTime`**, which does not match the design.

File(s) Affected

contracts/protocol/gauge/LendingGauge.sol #315-318

```
315       if (isKilled) {
316         // Stop distributing inflation as soon as killed
317         _st.rate = 0;
318       }
```

contracts/protocol/gauge/LendingGauge.sol #160-163

```
160       if (isKilled) {
161         // Stop distributing inflation as soon as killed
162         _st.rate = 0;
163       }
```

contracts/protocol/gauge/AbsGauge.sol #122-126

```
122             if (_st.prevFutureEpoch >= _prevWeekTime && _st.prevFutureEpoch < _weekTime) {
123               _integrateInvSupply += (_st.rate * _w * _allocation * (_st.prevFutureEpoch - _prevWeekTime)
124               _st.rate = _st.newRate;
125               _integrateInvSupply += (_st.rate * _w * _allocation * (_weekTime - _st.prevFutureEpoch)) /
126             } else {
```

Recommendation

Consider resetting **`newRate`** to zero if **`isKilled`** is true.

Alleviation Fixed

The development team resolved this issue by resetting `newRate` to 0 in the commit b3c0ba6068dd28c797d4e1b4a2fe1a9b1b70cfd6

3. The `kick` function may make users loss their reward

Medium risk Security Analyzer

The **`kick`** function can be called by anyone to update the arbitrary address's LT token reward:

```
function kick(address _addr) external {
  ...
  _userCheckpoint(_addr);
```

```
  }

  function _userCheckpoint(address _addr) internal {
    ...
    if (isKilled) {
      // Stop distributing inflation as soon as killed
      _st.rate = 0;
    }
    if (IHTokenRewards(hToken).totalSupply() != 0) {
      IHTokenRewards(hToken).checkpoint(_addr, _calRelativeWeightByAllocation(hToken), _st);
      IHTokenRewards(hToken).updateLiquidityLimit(_addr);
    }
    if (IVariableDebtTokenRewards(variableDebtToken).totalSupply() != 0) {
      IVariableDebtTokenRewards(variableDebtToken).checkpoint(_addr, _calRelativeWeightByAllocation(variableDebtToken), _st
      IVariableDebtTokenRewards(variableDebtToken).updateLiquidityLimit(_addr);
    }
    ...
  }
```

The point is that if the `isKilled` is set as 0 by the admin, the updated user would get 0 LT token reward.

File(s) Affected

contracts/protocol/gauge/LendingGauge.sol #242-257

```
242   function kick(address _addr) external {
243     uint256 _hTokenLast = IHTokenRewards(hToken).integrateCheckpointOf(_addr);
244     uint256 _variableDebtTokenLast = IVariableDebtTokenRewards(variableDebtToken).integrateCheckpointOf
245     uint256 _tVe = votingEscrow.userPointHistoryTs(_addr, votingEscrow.userPointEpoch(_addr));
246     uint256 _hTokenBalance = IHTokenRewards(hToken).lpBalanceOf(_addr);
247     uint256 _variableDebtTokenBalance = IHTokenRewards(hToken).lpBalanceOf(_addr);
248
249     require(votingEscrow.balanceOfAtTime(_addr, block.timestamp) == 0 || _tVe > _hTokenLast || _tVe > _
250     require(
251       IHTokenRewards(hToken).workingBalances(_addr) > (_hTokenBalance * 40) / 100 ||
252         IVariableDebtTokenRewards(variableDebtToken).workingBalances(_addr) > (_variableDebtTokenBalanc
253       'GP001'
254     );
255
256     _userCheckpoint(_addr);
257   }
```

contracts/protocol/gauge/LendingGauge.sol #315-324

```
315     if (isKilled) {
316       // Stop distributing inflation as soon as killed
317       _st.rate = 0;
318     }
319     if (IHTokenRewards(hToken).totalSupply() != 0) {
320       IHTokenRewards(hToken).checkpoint(_addr, _calRelativeWeightByAllocation(hToken), _st);
321       IHTokenRewards(hToken).updateLiquidityLimit(_addr);
322     }
323     if (IVariableDebtTokenRewards(variableDebtToken).totalSupply() != 0) {
324       IVariableDebtTokenRewards(variableDebtToken).checkpoint(_addr, _calRelativeWeightByAllocation(va
```

Recommendation

Consider checking if it is an intended design for anyone can call the `kick` function with an arbitrary address.

Alleviation Acknowledged

The development team acknowledged this finding.

Low risk (3)

1. Lack of checking the `failoverActive`

Low risk Security Analyzer

In the `HopeOracle` contract, the `OPERATOR_ROLE` is able to `activate` or `deactivate` a fallback price oracle:

```
function activateFailover(address asset) external override onlyRole(OPERATOR_ROLE) {
  _activateFailover(asset);
}

function deactivateFailover(address asset) external override onlyRole(OPERATOR_ROLE) {
  _deactivateFailover(asset);
}
```

Common sense is that the fallback price oracle should not be used if it is deactivated. However, in the final inner `else` branch of the `getAssetPrice` function, it lacks checking if the fallback oracle is deactivated or not:

```
function getAssetPrice(address asset) public view override returns (uint256) {
    ...
    } else if (priceData.failoverActive) {
      return _fallbackOracle.getAssetPrice(asset);
    } else {
      int256 price = priceData.source.latestAnswer();
      if (price > 0) {
        return uint256(price);
      } else {
        return _fallbackOracle.getAssetPrice(asset);
      }
    }
  }
```

The same case happened in the first `else if` branch of the `getAssetPrice` function:

```
function getAssetPrice(address asset) public view override returns (uint256) {
  PriceData storage priceData = assetsPriceDatas[asset];

  if (asset == BASE_CURRENCY) {
    return BASE_CURRENCY_UNIT;
  } else if (address(priceData.source) == address(0)) {
    return _fallbackOracle.getAssetPrice(asset);
  }
```

File(s) Affected

contracts/misc/HopeOracle.sol #155-172

```
155   function getAssetPrice(address asset) public view override returns (uint256) {
156     PriceData storage priceData = assetsPriceDatas[asset];
157
158     if (asset == BASE_CURRENCY) {
159       return BASE_CURRENCY_UNIT;
160     } else if (address(priceData.source) == address(0)) {
161       return _fallbackOracle.getAssetPrice(asset);
162     } else if (priceData.failoverActive) {
163       return _fallbackOracle.getAssetPrice(asset);
164     } else {
165       int256 price = priceData.source.latestAnswer();
166       if (price > 0) {
167         return uint256(price);
168       } else {
169         return _fallbackOracle.getAssetPrice(asset);
170       }
171     }
172   }
```

Recommendation

1. Consider avoiding using the fallback price oracle if it is deactivated.
2. Is using a deactivated fallback price oracle an intended design?

Alleviation Acknowledged

[HOPE]: We listen for prices returned by third-party data sources that are decoupled from the market price. When the price returned by the third-party data source is out of the market price and not zero, we (the risk control organization) will start a Failover for the Token and use the price of the Alternative Prophet. When the price returned by the third-party data source is in line with the market price, we will stop the Failover of the Token and use the price of the Primary Prophet (i.e. the price of the third-party data source).

2. The event mismatches the implementation

Low risk Security Analyzer

The **`executeMintToTreasury`** function mints HToken to the **`amountToMint`**, and emits the event **`MintedToTreasury`**:

```
function executeMintToTreasury(
  mapping(address => DataTypes.ReserveData) storage reservesData,
  address[] calldata assets
) external {
  ...
      IHToken(reserve.hTokenAddress).mintToTreasury(amountToMint, normalizedIncome);
      emit MintedToTreasury(assetAddress, amountToMint);
  ...
}
```

The **`MintedToTreasury`** event stands for the **`amountToMint`** amount of assets minted to the treasury account. Let's check the implementation of the **`mintToTreasury`** function:

```
function mintToTreasury(uint256 amount, uint256 index) external virtual override onlyPool {
  ...
  if (feeToVault != address(0) && feeToVaultPercent != 0) {
    uint256 amountToVault = amount.percentMul(feeToVaultPercent);
    IERC20(_underlyingAsset).safeTransfer(feeToVault, amountToVault);
    _mintScaled(address(POOL), _treasury, amount - amountToVault, index);
  }
  ...
}
```

The `mintToTreasury` function only mints `amount - amountToVault` amount of assets to the treasury account if `feeToVault` and `feeToVaultPercent` are non-zero values, the `amountToVault` amount of underly asset is indeed transferred to the `feeToVault`.

File(s) Affected

contracts/protocol/libraries/logic/PoolLogic.sol #104-111

```
104       if (accruedToTreasury != 0) {
105         reserve.accruedToTreasury = 0;
106         uint256 normalizedIncome = reserve.getNormalizedIncome();
107         uint256 amountToMint = accruedToTreasury.rayMul(normalizedIncome);
108         IHToken(reserve.hTokenAddress).mintToTreasury(amountToMint, normalizedIncome);
109
110         emit MintedToTreasury(assetAddress, amountToMint);
111       }
```

contracts/protocol/tokenization/HToken.sol #105-109

```
105     if (feeToVault != address(0) && feeToVaultPercent != 0) {
106       uint256 amountToVault = amount.percentMul(feeToVaultPercent);
107       IERC20(_underlyingAsset).safeTransfer(feeToVault, amountToVault);
108       _mintScaled(address(POOL), _treasury, amount - amountToVault, index);
109     } else {
```

Recommendation

Consider emitting an event in the `mintToTreasury` function if `feeToVault` and `feeToVaultPercent` are non-zero values for asset tracking.

Alleviation Fixed

The development team resolved this issue by emitting a new event in commit f53383c6d059f83e40b70394aaf027a55cc17ab1

3. Gas optimization

Low risk Security Analyzer

In the `_checkpoint` function, the storage variable `_integrateInvSupply` is read at least 3 times and written at least 1 time.

Note that the `for` loop may increase the read times and write times of the `_integrateInvSupply`.

We can declare a new memory variable and cache the value and calculation result for the `_integrateInvSupply` to save gas.

File(s) Affected

contracts/protocol/gauge/AbsGauge.sol #104-141

```
104   function _checkpoint(
105     address _addr,
106     uint256 _allocation,
107     DataTypes.CheckPointParameters memory _st
108   ) internal {
109     if (block.timestamp > _st.periodTime) {
110       uint256 _workingSupply = workingSupply;
111       uint256 _prevWeekTime = _st.periodTime;
112       uint256 _weekTime = Math.min(((_st.periodTime + _WEEK) / _WEEK) * _WEEK, block.timestamp);
113       for (uint256 i; i < 500; i++) {
114         uint256 _dt = _weekTime - _prevWeekTime;
115         uint256 _preWeekTimeRound = (_prevWeekTime / _WEEK) * _WEEK;
116         uint256 _w = historyGaugeRelativeWeight[_preWeekTimeRound];
117         if (_w == 0) {
118           _w = controller.gaugeRelativeWeight(address(lendingGauge), _preWeekTimeRound);
119           historyGaugeRelativeWeight[_preWeekTimeRound] = _w;
120         }
121         if (_workingSupply > 0) {
122           if (_st.prevFutureEpoch >= _prevWeekTime && _st.prevFutureEpoch < _weekTime) {
123             _integrateInvSupply += (_st.rate * _w * _allocation * (_st.prevFutureEpoch - _prevWeekTime)
124             _st.rate = _st.newRate;
125             _integrateInvSupply += (_st.rate * _w * _allocation * (_weekTime - _st.prevFutureEpoch)) /
126           } else {
127             _integrateInvSupply += (_st.rate * _w * _allocation * _dt) / _workingSupply / WadRayMath.RA
128           }
129         }
130         if (_weekTime == block.timestamp) {
131           break;
132         }
133         _prevWeekTime = _weekTime;
134         _weekTime = Math.min(_weekTime + _WEEK, block.timestamp);
135       }
136     }
137     uint256 _workingBalance = workingBalances[_addr];
138     integrateFraction[_addr] += (_workingBalance * (_integrateInvSupply - integrateInvSupplyOf[_addr]))
139     integrateInvSupplyOf[_addr] = _integrateInvSupply;
140     integrateCheckpointOf[_addr] = block.timestamp;
141   }
```

Recommendation

Consider reducing the number of times of read and write the storage variable in functions when reading and writing the storage variables throughout the whole project.

Alleviation Acknowledged

The development team acknowledged this finding.

Informational (4)

1. Unused Parameter

Informational Security Analyzer

The parameter `oldTotalSupply` in the `_mint` function and `_burn` function are unused:

```
function _mint(
  address account,
  uint256 amount,
  uint256 oldTotalSupply
) internal {
  uint128 castAmount = amount.toUint128();
  uint128 oldAccountBalance = _userState[account].balance;
  _userState[account].balance = oldAccountBalance + castAmount;
}

  function _burn(
  address account,
  uint256 amount,
  uint256 oldTotalSupply
) internal {
  uint128 castAmount = amount.toUint128();
  uint128 oldAccountBalance = _userState[account].balance;
  _userState[account].balance = oldAccountBalance - castAmount;
}
```

File(s) Affected

contracts/protocol/tokenization/StableDebtToken.sol #320-344

```
320   function _mint(
321     address account,
322     uint256 amount,
323     uint256 oldTotalSupply
324   ) internal {
325     uint128 castAmount = amount.toUint128();
326     uint128 oldAccountBalance = _userState[account].balance;
327     _userState[account].balance = oldAccountBalance + castAmount;
328   }
329
330   /**
331    * @notice Burns stable debt tokens of a user
332    * @param account The user getting his debt burned
333    * @param amount The amount being burned
334    * @param oldTotalSupply The total supply before the burning event
335    */
336   function _burn(
337     address account,
338     uint256 amount,
339     uint256 oldTotalSupply
340   ) internal {
341     uint128 castAmount = amount.toUint128();
342     uint128 oldAccountBalance = _userState[account].balance;
343     _userState[account].balance = oldAccountBalance - castAmount;
344   }
```

Recommendation

Consider removing the unused parameters and no need to pass them.

Alleviation Fixed

The development team resolved this issue by removing unnecessary parameters in commit 65a07f7ee73dbd5ec2e9cc55b3f7fb8b29aae69c

2. Missing emit event

Informational Security Analyzer

The key state updates are recommended to emit events to track their state.

File(s) Affected

contracts/protocol/gauge/AbsGauge.sol #53-59

```
53   function _setLendingGauge(address _lendingPoolGuageAddr) internal {
54     lendingGauge = ILendingGauge(_lendingPoolGuageAddr);
55     if (_lendingPoolGuageAddr != address(0)) {
56       controller = lendingGauge.controller();
57       votingEscrow = lendingGauge.votingEscrow();
58     }
59   }
```

contracts/protocol/pool/Pool.sol #572-577

```
572   function setLendingGauge(address asset, address lendingGauge) external onlyPoolAdmin {
573     require(asset != address(0), Errors.ZERO_ADDRESS_NOT_VALID);
574     require(_reserves[asset].id != 0 || _reservesList[0] == asset, Errors.ASSET_NOT_LISTED);
575     IHTokenRewards(_reserves[asset].hTokenAddress).setLendingGauge(lendingGauge);
576     IVariableDebtTokenRewards(_reserves[asset].variableDebtTokenAddress).setLendingGauge(lendingGauge);
577   }
```

contracts/protocol/gauge/LendingGauge.sol #233-235

```
233   function setKilled(bool _isKilled) external onlyPoolAdmin {
234     isKilled = _isKilled;
235   }
```

Recommendation

Consider emitting events for key state updates.

Alleviation Acknowledged

The development team acknowledged this finding.

3. The potential inconsistent between `lendingGauge` and `allLendingGauges`

Informational Security Analyzer

The `createLendingGauge` function creates lending gauge for the underlying asset, stores mapping from underlying asset to lending gauge, and stores all the lending gauges into `allLendingGauges`.

If the pool admin invokes `setLendingGaugeImplementation` function and `createLendingGauge` function to update the implementation for an underlying asset to replace its old lending gauge, the `allLendingGauges` would contain lending gauges for the same underlying asset, which would result in the length of `allLendingGauges` being greater than it of `lendingGauge`, because the `createLendingGauge` function does not check if the lending gauge was created before or not.

File(s) Affected

contracts/protocol/gauge/GaugeFactory.sol #53-61

```
53   function createLendingGauge(address _underlyingAsset) external onlyRole(OPERATOR_ROLE) returns (addres
54     require(_underlyingAsset != address(0), Errors.ZERO_ADDRESS_NOT_VALID);
55     bytes32 salt = keccak256(abi.encodePacked(_underlyingAsset));
56     lendingGaugeAddress = Clones.cloneDeterministic(lendingGaugeImplementation, salt);
57     ILendingGauge(lendingGaugeAddress).initialize(pool, minter, votingEscrow, _underlyingAsset);
58     lendingGauge[_underlyingAsset] = lendingGaugeAddress;
59     allLendingGauges.push(lendingGaugeAddress);
60     emit LendingGaugeCreated(address(_addressesProvider), _underlyingAsset, lendingGaugeAddress, allLend
61   }
```

Recommendation

Consider checking if it is an intended design, and updating the implementation if necessary.

Alleviation Acknowledged

The development team acknowledged this finding.

4. The return value of `_getAllocationByUtilizationRate` function might be misleading under the edge case

Informational Security Analyzer

The `_getAllocationByUtilizationRate` calculates the debt token allocation ratio based on the fund utilization rate.

It is ok to return 0 if `_utilizationRate` is zero:

```
function _getAllocationByUtilizationRate(uint256 _utilizationRate) internal view returns (uint256) {
  require(phases.length > 0, Errors.PHASES_NOT_DEFINED);
  if (_utilizationRate == 0) {
    return 0;
  }
  ...
}
```

Then, if the `_utilizationRate` is not zero, the `for` loop will iterate all the items of the `phases` and find the right phase to calculate the debt token allocation ratio.

```
function _getAllocationByUtilizationRate(uint256 _utilizationRate) internal view returns (uint256) {
  ...
  for (uint256 i = 0; i < phases.length; i++) {
    if (_utilizationRate > phases[i].start && _utilizationRate <= phases[i].end) {
      int256 _borrowAllocation = (phases[i].k * _utilizationRate.toInt256()) / WadRayMath.RAY.toInt256() + phases[i].b.to
      require(_borrowAllocation >= 0, Errors.MUST_BE_NON_NEGATIVE);
      return _borrowAllocation.toUint256();
    }
  }
  ...
}
```

But, for the edge case that the `for` loop fails to find the right phase if the `_utilizationRate` is not zero, the `_getAllocationByUtilizationRate` function still returns 0, which is the same as the case that the `_utilizationRate` is zero, as a result, the caller is unable to distinct what is a return value of zero stands for.

If the `_utilizationRate` is not zero and a set of wrong phases is set would result in a zero being returned from the `_getAllocationByUtilizationRate` function, which is harmful to the system.

File(s) Affected

contracts/protocol/gauge/LendingGauge.sol #278-291

```
278   function _getAllocationByUtilizationRate(uint256 _utilizationRate) internal view returns (uint256) {
279     require(phases.length > 0, Errors.PHASES_NOT_DEFINED);
280     if (_utilizationRate == 0) {
281       return 0;
282     }
283     for (uint256 i = 0; i < phases.length; i++) {
284       if (_utilizationRate > phases[i].start && _utilizationRate <= phases[i].end) {
285         int256 _borrowAllocation = (phases[i].k * _utilizationRate.toInt256()) / WadRayMath.RAY.toInt25
286         require(_borrowAllocation >= 0, Errors.MUST_BE_NON_NEGATIVE);
287         return _borrowAllocation.toUint256();
288       }
289     }
290     return 0;
291   }
```

Recommendation

Consider reverting if the `_utilizationRate` is not zero and the `for` loop fails to find the right phase.

Alleviation Acknowledged

The development team acknowledged this finding.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without MetaTrust's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts MetaTrust to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. MetaTrust's position is that each company and individual are responsible for their own due diligence and continuous security. MetaTrust's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by MetaTrust is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS Security Assessment AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, MetaTrust HEREBY DISCLAIMS ALL WARRANTIES,

WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, MetaTrust SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, MetaTrust MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, MetaTrust PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER MetaTrust NOR ANY OF MetaTrust'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. MetaTrust WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT MetaTrust'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING Security Assessment MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST MetaTrust WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF MetaTrust CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST MetaTrust WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.