Light-Ecosystem-lend-core

Smart Contract Security Audit

V1.0

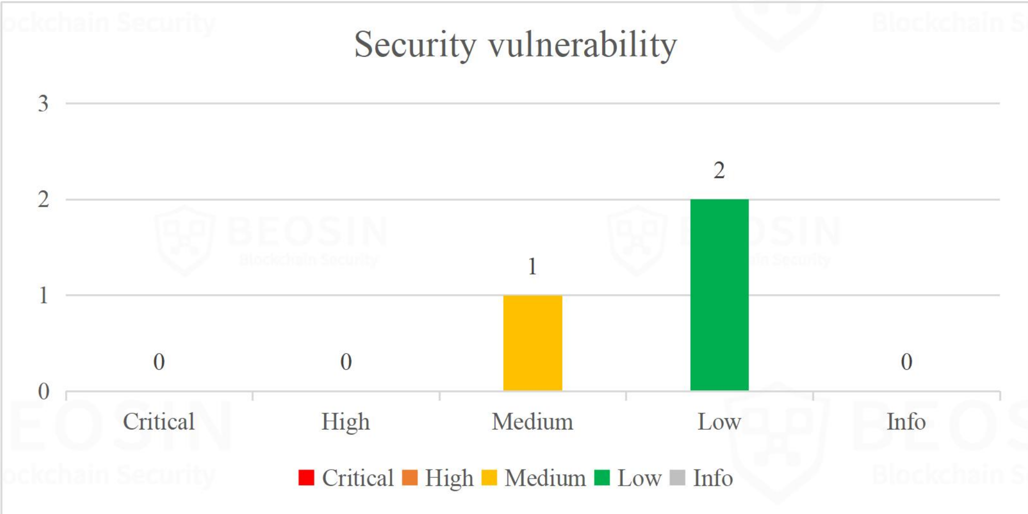
No. 202307191800

Jul 19th, 2023

Contents

Summary of Audit Results

After auditing, 1 Medium and 2 Low risk were identified in the Light-Ecosystem-lend-core project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project:



- **Project Description:**

1. Business overview

The main functions of this part of lend-core code are mortgage, lending, cross-chain bridge, lightning lending, liquidation and other functions. Users will mint corresponding HToken and VariableDebtToken when pledging assets and lending, and the contract will provide LT incentives to users according to the utilization rate of funds (the reward share of HToken and VariableDebtToken is always 1). Different fund utilization rates correspond to different LT incentives, which depend on the setting of the owner. For example, at the beginning, users are encouraged to borrow. In the case of low fund utilization rate, the reward of lending users is higher than that of mortgaged users; on the contrary, in the case of high fund utilization rate, the corresponding LT incentive share of mortgaged users accounts for a large proportion.

1 Overview

1.1 Project Overview

Project Name	Light-Ecosystem-lend-core
Platform	Ethereum
GitHub	https://github.com/Light-Ecosystem/lend-core/tree/audit
Commit	97eb3ef81db94b52616230ad036dee58192a907c 82b443d7b507b4a6f1024c2df5302ab9556d7991 b3c0ba6068dd28c797d4e1b4a2fe1a9b1b70cfd6 3fd921a60f5d5d267bb274c8c998c51d45436177

1.2 Audit Overview

Audit work duration: June 20, 2023 – July 19, 2023

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team.

2 Findings

Index	Risk description	Severity level	Status
Light-Ecosystem-lend-core-1	Users can still earn rewards after turning off rewards	Medium	Fixed
Light-Ecosystem-lend-core-2	Wrong use of Htoken to get Debttoken balance	Low	Fixed
Light-Ecosystem-lend-core-3	Missing update allocation in *mintToTreasury* function	Low	Fixed

Finding Details:

[Light-Ecosystem-lend-core-1] Users can still earn rewards after turning off rewards

Severity Level	Medium
Type	Business Security
Lines	LendingGauge.sol#L156-159
Description	The *hvCheckpoint* function is used to settle rewards, in the *hvCheckpoint* function the state of isKilled is determined, if isKilled is true, it will make the settled reward zero. Since the reward settlement will be across epochs, the development here uses _st.rate and _st.newRate parameters to calculate the reward. Since after turning on isKilled to true, it doesn't take into account that _st.newRate still has a value, which will lead to inter-epoch cases and still allow for reward settlement.

```
139 function hvCheckpoint(address _addr) public override {
140   DataTypes.CheckPointParameters memory _st;
141   _st.period = period;
142   _st.periodTime = periodTimestamp[_st.period];
143   _st.rate = inflationRate;
144   _st.newRate = _st.rate;
145   _st.prevFutureEpoch = futureEpochTime;
146   if (_st.prevFutureEpoch <= block.timestamp) {
147     futureEpochTime = ltToken.futureEpochTimeWrite();
148     _st.newRate = ltToken.rate();
149     inflationRate = _st.newRate;
150   }
151   uint256 _weekTime = (block.timestamp / _WEEK) * _WEEK;
152   if (!checkedGauge[_weekTime]) {
153     checkedGauge[_weekTime] = true;
154     controller.checkpointGauge(address(this));
155   }
156   if (isKilled) {
157     // Stop distributing inflation as soon as killed
158     _st.rate = 0;
159   }
160   if (IHTokenRewards(hToken).totalSupply() != 0) {
161     IHTokenRewards(hToken).checkpoint(_addr, _calRelativeWeightByAllocation(hToken), _st);
162   }
163   if (IVariableDebtTokenRewards(variableDebtToken).totalSupply() != 0) {
164     IVariableDebtTokenRewards(variableDebtToken).checkpoint(_addr, _calRelativeWeightByAllocation(variableDebtToken), _st);
165   }
166   _st.period += 1;
167   period = _st.period;
168   periodTimestamp[_st.period] = block.timestamp;
169 }
170
```

Figure 1 Screenshot of *hvCheckpoint* function code(Unfixed)

```solidity
function _checkpoint(
  address _addr,
  uint256 _allocation,
  DataTypes.CheckPointParameters memory _st
) internal {
  if (block.timestamp > _st.periodTime) {
    uint256 _workingSupply = workingSupply;
    uint256 _prevWeekTime = _st.periodTime;
    uint256 _weekTime = Math.min(((_st.periodTime + _WEEK) / _WEEK) * _WEEK, block.timestamp);
    for (uint256 i; i < 500; i++) {
      uint256 _dt = _weekTime - _prevWeekTime;
      uint256 _preWeekTimeRound = (_prevWeekTime / _WEEK) * _WEEK;
      uint256 _w = historyGaugeRelativeWeight[_preWeekTimeRound];
      if (_w == 0) {
        _w = controller.gaugeRelativeWeight(address(lendingGauge), _preWeekTimeRound);
        historyGaugeRelativeWeight[_preWeekTimeRound] = _w;
      }
      if (_workingSupply > 0) {
        if (_st.prevFutureEpoch >= _prevWeekTime && _st.prevFutureEpoch < _weekTime) {
          _integrateInvSupply += (_st.rate * _w * _allocation * (_st.prevFutureEpoch - _prevWeekTime)) / _workingSupply / WadRayMath.RAY;
          _st.rate = _st.newRate;
          _integrateInvSupply += (_st.rate * _w * _allocation * (_weekTime - _st.prevFutureEpoch)) / _workingSupply / WadRayMath.RAY;
        } else {
          _integrateInvSupply += (_st.rate * _w * _allocation * _dt) / _workingSupply / WadRayMath.RAY;
        }
      }
      if (_weekTime == block.timestamp) {
        break;
      }
      _prevWeekTime = _weekTime;
      _weekTime = Math.min(_weekTime + _WEEK, block.timestamp);
    }
  }
  uint256 _workingBalance = workingBalances[_addr];
  integrateFraction[_addr] += (_workingBalance * (_integrateInvSupply - integrateInvSupplyOf[_addr])) / 10**18;
  integrateInvSupplyOf[_addr] = _integrateInvSupply;
  integrateCheckpointOf[_addr] = block.timestamp;
}
```

Figure 2 Screenshot of _*checkpoint* function code

Recommendations	It is recommended to set _st.newRate to zero.
Status	Fixed. Fix the commit as follows: https://github.com/Light-Ecosystem-lend-core/lend-core/commit/b3c0ba6068dd28c797d4e1b4a2fe1a9b1b70cfd6.

[Light-Ecosystem-lend-core-2] Wrong use of Htoken to get Debttoken balance

Severity Level	**Low**
Type	Business Security
Lines	LendingGauge.sol#L243
Description	The *kick* function in the LendingGauge contract, when used to get the _variableDebtTokenBalance value for the specified address, incorrectly uses the Htoken contract to get it, which will result in an incorrect implementation of the *kick* function.

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

Figure 3 Screenshot of *kick* function code(Unfixed)

Recommendations	Suggest to change htoken to variableDebtToken address.
Status	Fixed. Fix the commit as follows: https://github.com/Light-Ecosystem-lend-core/lend-core/commit/82b443d7b507b4a6f1024c2df5302ab9556d7991.

[Light-Ecosystem-lend-core-3] Missing update allocation in *mintToTreasury* function

Severity Level	Low
Type	Business Security
Lines	PoolLogic.sol#L88-113
Description	In the *updateAllocation* function of the LendingGauge.sol contract, it is understood that the quantity of underlyingAsset in the Htoken contract will affect the Allocation. However, in the *executeMintToTreasury* function, even though the *mintToTreasury* function is called, the Allocation is not updated. This will lead to an inaccurate Allocation because when calling the *mintToTreasury* function in the Htoken contract, a portion of the quantity of _underlyingAsset is transferred to the feeToVault address.

```
function mintToTreasury(uint256 amount, uint256 index) external virtual override onlyPool {
  if (amount == 0) {
    return;
  }

  address feeToVault = POOL.getFeeToVault();
  uint256 feeToVaultPercent = POOL.getFeeToVaultPercent();
  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);
  }
}
```

Figure 4 Screenshot of *mintToTreasury* function code

```
*/
function updateAllocation() external override 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;
  if (availableLiquidityPlusDebt == 0) {
    borrowAllocation = 0;
    return false;
  }
  borrowAllocation = _getAllocationByUtilizationRate(totalDebt.rayDiv(availableLiquidityPlusDebt));
  return true;
}
```

Figure 5 Screenshot of *updateAllocation* function code

```
function executeMintToTreasury(
  mapping(address => DataTypes.ReserveData) storage reservesData,
  address[] calldata assets
) external {
  for (uint256 i = 0; i < assets.length; i++) {
    address assetAddress = assets[i];

    DataTypes.ReserveData storage reserve = reservesData[assetAddress];

    // this cover both inactive reserves and invalid reserves since the flag will be 0 for both
    if (!reserve.configuration.getActive()) {
      continue;
    }

    uint256 accruedToTreasury = reserve.accruedToTreasury;

    if (accruedToTreasury != 0) {
      reserve.accruedToTreasury = 0;
      uint256 normalizedIncome = reserve.getNormalizedIncome();
      uint256 amountToMint = accruedToTreasury.rayMul(normalizedIncome);
      IHToken(reserve.hTokenAddress).mintToTreasury(amountToMint, normalizedIncome);

      emit MintedToTreasury(assetAddress, amountToMint);
    }
  }
}
```

Figure 6 Screenshot of *executeMintToTreasury* function code(Unfixed)

Recommendations	It is recommended to update Allocation in the *mintToTreasury* function.
Status	Fixed. Fix the commit as follows: https://github.com/Light-Ecosystem-lend-core/lend-core/commit/3fd921a60f5d5d267bb274c8c998c51d45436177.

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itsLight-Ecosystem-lend-core, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

service@beosin.com