

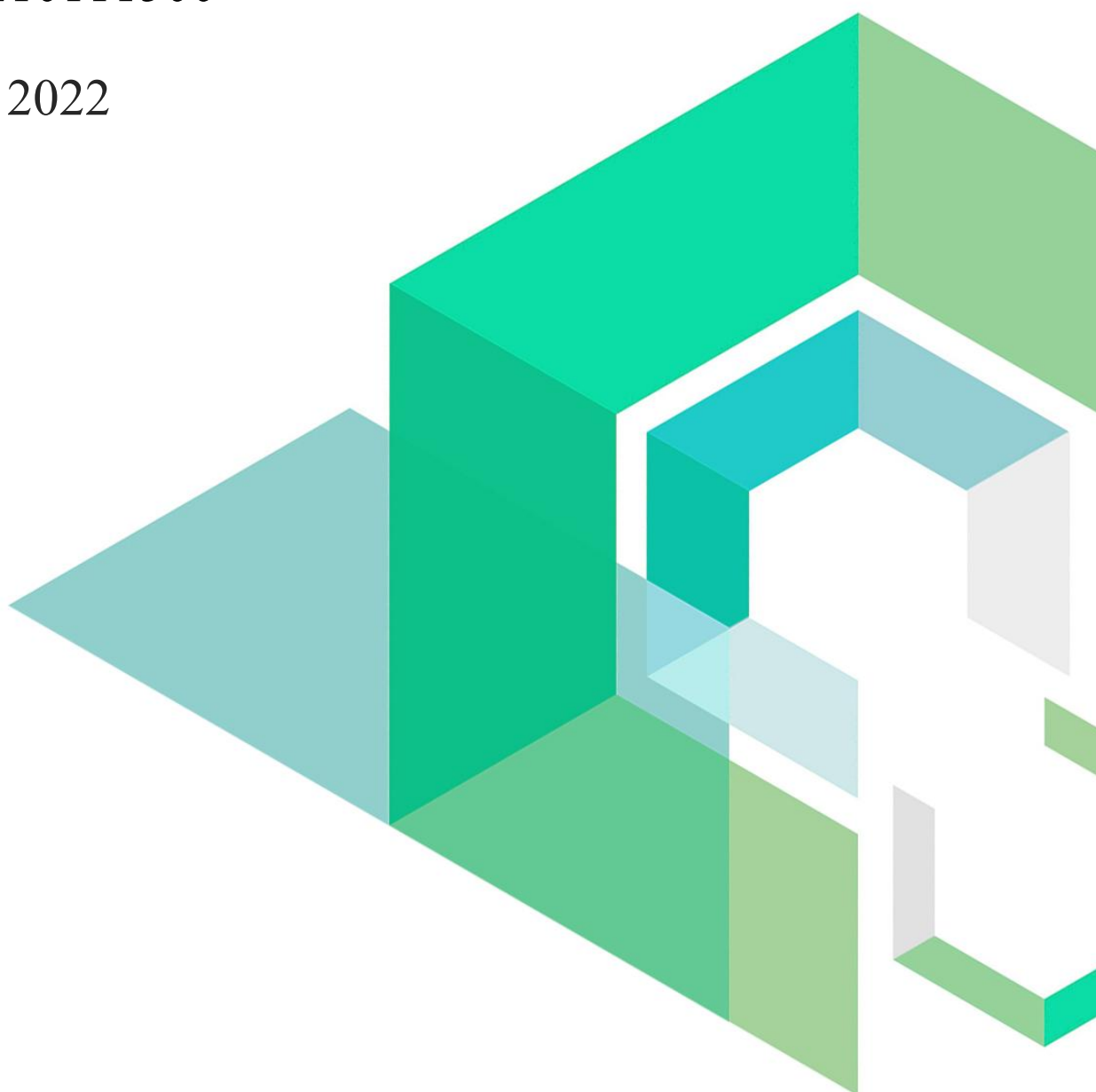
LeverFi

Smart Contract Security Audit

V1.3

No. 202210111500

Oct 11th, 2022



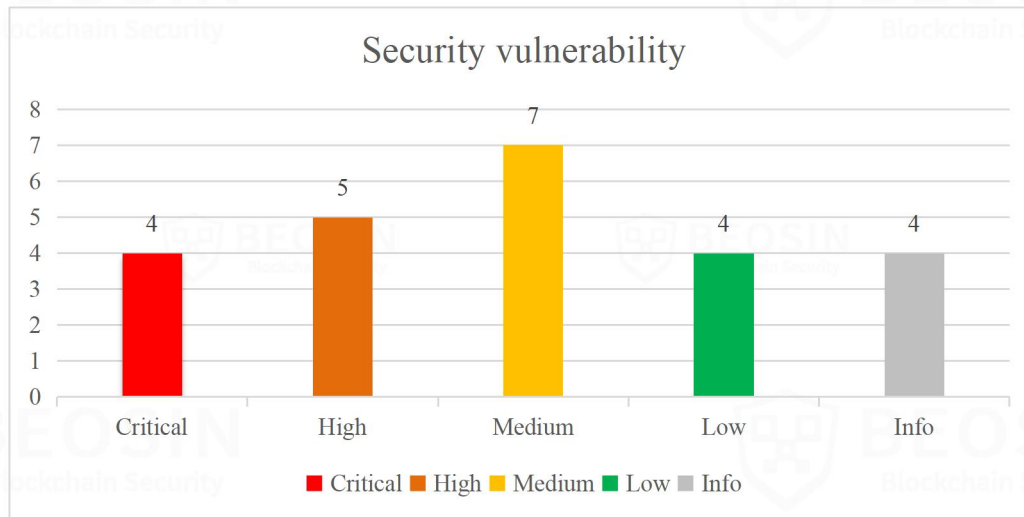
Contents

Summary of audit results	1
1 Overview	4
1.1 Project Overview	4
1.2 Audit Overview	4
2 Findings	5
[LeverFi-1] The <i>initialize</i> function lacks permission check	6
[LeverFi-2] The <i>swapPosition</i> function lacks permission check	8
[LeverFi-3] The <i>swap</i> function is improperly designed	10
[LeverFi-4] The <i>withdrawLiquidationWalletLong</i> function is improperly designed	12
[LeverFi-5] DoS attack	14
[LeverFi-6] The <i>_calculateRewards</i> function design flaw	16
[LeverFi-7] The <i>_depositReserve</i> function is improperly designed	18
[LeverFi-8] The <i>configureAsset</i> function is improperly designed	20
[LeverFi-9] Clearing mechanism error	21
[LeverFi-10] The <i>getNormalizedDebt</i> function is improperly designed	22
[LeverFi-11] Implementation flaws in the <i>reinvestReserveSupply</i> and <i>reinvestCollateralSupply</i> functions	24
[LeverFi-12] Unsafe call	26
[LeverFi-13] The <i>configureReserve</i> function design flaw	27
[LeverFi-14] Centralization risk	30
[LeverFi-15] The <i>_repayShort</i> function is improperly designed	32
[LeverFi-16] The <i>ExecutewithdrawReserve</i> function lacks update percentageray parameters	34
[LeverFi-17] Improperly designed <i>_depositCollateral</i> function	36
[LeverFi-18] The <i>validateTrade</i> function check error	37

[LeverFi-19] The <i>configureCollateral</i> function cannot modify reinvestment	38
[LeverFi-20] Improper design when settlement user awards	40
[LeverFi-21] lack of judgment on value	41
[LeverFi-22] The <i>getUserLiquidity</i> function design flaw	43
[LeverFi-23] Lack of judgment on whether to add assets	44
[LeverFi-24] Admin permission not initialized	46
3 Appendix	47
3.1 Vulnerability Assessment Metrics and Status in Smart Contracts	47
3.2 Audit Categories	49
3.3 Disclaimer	51
3.4 About BEOSIN	52

Summary of audit results

After auditing, 4 Critical-risk, 5 High-risk, 7 Medium-risk, 4 Low-risk and 4 Info-risk items were identified in the LeverFi project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:



*Notes:

● Risk Description:

1. This project only audits the internal code of the project, and does not audit the interactive contract. Audited contracts include UserConfiguration.sol, CollateralLogic.sol, CollateralPoolLogic.sol, GeneralLogic.sol, LiquidationLogic.sol, PositionLogic.sol, ReserveLogic.sol, ReservePoolLogic.sol, TradeLogic.sol, ValidationLogic.sol, HelpersLogic.sol, ChainLinkPriceOracleGetter.sol, CurveLPOracleGetter.sol, MockPriceOracle.sol, StargateLPOracleGetter.sol, TriCryptoOracle.sol, LedgerStorage.sol, AaveReinvestmentLogic.sol, ConvexReinvestmentLogic.sol, StargateReinvestmentLogic.sol, ReinvestmentProxy.sol, ZeroexSwapAdapter.sol, DataTypes.sol, UnwrapLp.sol, BonusPool.sol, Configurator.sol, UserData.sol and Ledger.sol
2. There is test code in the project, please do not use the test code, such as MockPriceOracle.sol, etc.
3. This audit report is only for the current code, but the business logic contract of the current project can be upgraded, and the code after the upgrade cannot be determined. After the upgrade, the risk of loss of funds and data may be introduced. When users interact with this project, they need to pay attention to whether the upgraded logic contract is consistent with the audit code.
4. Due to problems with CurveOracle prophecy machine, it can lead to price manipulation, which can result in loss of contract funds. Details can be found at: <https://chainsecurity.com/curve-lp-oracle-manipulation->

post-mortem/. Therefore a modifier is proposed to be added, which restricts contract calls using `tx.origin==msg.sender` in the modifier, which will prevent the attack from happening.

- **Project Description:**

- 1. Business overview**

The LeverFi project is a revenue aggregation project that allows liquidity providers to provide funds to earn interest. Liquidity providers deposit funds into lending pools, and lenders earn interest from borrowing leveraged traders. Idle assets in lending pools that are not used (unused liquidity) by traders will be deployed to other DeFi protocols for yield. When a trader deposits BTC, ETH, Curve-LP, Uni-LP and other collaterals in the collateral pool, the contract will deposit the collateral into the DeFi protocol to obtain benefits, while allowing traders to perform leveraged transactions. Leveraged trades are only entered, stored and settled within the LeverFi platform. If a trade loses close to the value of the deposited collateral, the liquidator system will make a margin call to the trader.

1 Overview

1.1 Project Overview

Project Name	LeverFi
Platform	Ethereum
Github	https://github.com/LeverFi/main-contracts
Commit	ed904fe9caabab160fdaed965094613236ba9308(original) 19e22c2bd832db1cc85eedbbe45855be9db042e(fixed)

1.2 Audit Overview

Audit work duration: August 4, 2022 – October 11, 2022

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
LeverFi-1	The <i>initialize</i> function lacks permission check	Critical	Fixed
LeverFi-2	The <i>swapPosition</i> function lacks permission check	Critical	Fixed
LeverFi-3	The <i>swap</i> function is improperly designed	Critical	Fixed
LeverFi-4	The <i>withdrawLiquidationWalletLong</i> function is improperly designed	Critical	Fixed
LeverFi-5	DoS attack	High	Fixed
LeverFi-6	The <i>_calculateRewards</i> function design flaw	High	Fixed
LeverFi-7	The <i>_depositReserve</i> function is improperly designed	High	Fixed
LeverFi-8	The <i>configureAsset</i> function is improperly designed	High	Fixed
LeverFi-9	Clearing mechanism error	High	Fixed
LeverFi-10	The <i>getNormalizedDebt</i> function is improperly designed	Medium	Fixed
LeverFi-11	Implementation flaws in the <i>reinvestReserveSupply</i> and <i>reinvestCollateralSupply</i> functions	Medium	Fixed
LeverFi-12	Unsafe call	Medium	Fixed
LeverFi-13	The <i>configureReserve</i> function design flaw	Medium	Fixed
LeverFi-14	Centralization risk	Medium	Fixed
LeverFi-15	The <i>_repayShort</i> function is improperly designed	Medium	Fixed
LeverFi-16	The <i>ExecutewithdrawReserve</i> function lacks update percentageray parameters	Medium	Fixed
LeverFi-17	Improperly designed <i>_depositCollateral</i> function	Low	Fixed
LeverFi-18	The <i>validateTrade</i> function check error	Low	Fixed
LeverFi-19	The <i>configureCollateral</i> function cannot modify reinvestment	Low	Fixed
LeverFi-20	Improper design when settlement user awards	Low	Fixed
LeverFi-21	lack of judgment on value	Info	Fixed
LeverFi-22	The <i>getUserLiquidity</i> function design flaw	Info	Fixed
LeverFi-23	Lack of judgment on whether to add assets	Info	Fixed
LeverFi-24	Admin permission not initialized	Info	Fixed

Finding Details:

[LeverFi-1] The *initialize* function lacks permission check

Severity Level	Critical
Type	Business Security
Lines	AaveReinvestmentLogic.sol,ConvexReinvestmentLogic.sol#L47,79
Description	In the initialize of the ConvexReinvestmentLogic and AaveReinvestmentLogic contracts, there is no permission to check and use the Initializable contract, which will cause anyone to call this function to modify it, thereby withdrawing the funds in the contract.

```

37
38  function initialize(
39      address asset_,
40      address receipt_,
41      address platform_,
42      address[] memory,
43      address treasury_,
44      address ledger_,
45      uint256,
46      bytes memory
47  ) public {
48      asset = asset_;
49      addressStorage[RECEIPT] = receipt_;
50      addressStorage[PLATFORM] = platform_;
51      treasury = treasury_;
52      ledger = ledger_;
53
54      // Fees does not applied to this type of reinvestment (auto-accruing reward).
55      feeMantissa = 0;
56  }

```

Figure 1 Source code of AaveReinvestmentLogic contract(Unfixed)

```

69  /**
70  function initialize(
71      address asset_,
72      address receipt_,
73      address platform_,
74      address[] memory rewards_,
75      address treasury_,
76      address ledger_,
77      uint256 feeMantissa_,
78      bytes memory data
79  ) public {
80      asset = asset_;
81      addressStorage[RECEIPT] = receipt_;
82      addressStorage[PLATFORM] = platform_;
83      treasury = treasury_;
84      ledger = ledger_;
85      feeMantissa = feeMantissa_;
86
87      (uint256 poolId_, address rewardPool_) = abi.decode(data, (uint256, address));
88      uintStorage[POOL_ID] = poolId_;
89      addressStorage[REWARD_POOL] = rewardPool_;
90
91      uintStorage[REWARD_LENGTH] = rewards_.length;
92
93      for (uint256 i = 0; i < rewards_.length; i++) {
94          setRewards(i, RewardConfig(i, rewards_[i], 0, 0));
95      }
96  }

```

Figure 2 Source code of ConvexReinvestmentLogic contract(Unfixed)

Recommendations It is recommended to use the Initializable contract.

Status

Fixed.

```
function initialize(  
    address asset_,  
    address receipt_,  
    address platform_,  
    address[] memory,  
    address treasury_,  
    address ledger_,  
    uint256,  
    bytes memory  
) external initializer onlyOwner {  
    asset = asset_;  
    addressStorage[RECEIPT] = receipt_;  
    addressStorage[PLATFORM] = platform_;  
    treasury = treasury_;  
    ledger = ledger_;  
  
    // Fees does not applied to this type of reinvestment (auto-accruing reward).  
    feeMantissa = 0;  
}
```

Figure 3 Source code of AaveReinvestmentLogic contract(Fixed)

```

70  */
71  function initialize(
72      address asset_,
73      address receipt_,
74      address platform_,
75      address[] memory rewards_,
76      address treasury_,
77      address ledger_,
78      uint256 feeMantissa_,
79      bytes memory data
80  ) external initializer onlyOwner {
81      asset = asset_;
82      addressStorage[RECEIPT] = receipt_;
83      addressStorage[PLATFORM] = platform_;
84      treasury = treasury_;
85      ledger = ledger_;
86      feeMantissa = feeMantissa_;
87
88      (uint256 poolId_, address rewardPool_) = abi.decode(data, (uint256, address));
89      uintStorage[POOL_ID] = poolId_;
90      addressStorage[REWARD_POOL] = rewardPool_;
91
92      uintStorage[REWARD_LENGTH] = rewards_.length;
93
94      for (uint256 i = 0; i < rewards_.length; i++) {
95          setRewards(i, RewardConfig(i, rewards_[i], 0, 0));
96      }
97  }

```

Figure 4 Source code of ConvexReinvestmentLogic contract(Fixed)

[LeverFi-2] The *swapPosition* function lacks permission check

Severity Level	Critical
Type	General Vulnerability
Lines	Ledger.sol#L1147
Description	The <i>swapPosition</i> function lacks permission verification. Anyone can execute this function and use the contract's tokens to swap, which will result in a loss of contract funds.

```

1146
1147     function swapPosition(address assetIn, address assetOut, uint256 amount, bytes memory data) external {
1148         TradeLogic.executeTrade(
1149             _reserveConfig,
1150             assetConfig,
1151             _userPosition[LIQUIDATION_WALLET],
1152             userConfig[LIQUIDATION_WALLET],
1153             DataTypes.ExecuteSwapParams(
1154                 LIQUIDATION_WALLET,
1155                 treasury,
1156                 assetIn,
1157                 assetOut,
1158                 amount,
1159                 tradeFeeMantissa,
1160                 0,
1161                 0,
1162                 0,
1163                 data,
1164                 false,
1165                 false
1166             )
1167         );
1168     }
1169

```

Figure 5 Source code of *swapPosition* function(Unfixed)

```

80  */
81  function executeTrade(
82      mapping(address => DataTypes.ReserveConfig) storage reserveConfig,
83      mapping(address => DataTypes.AssetConfig) storage assetConfig,
84      mapping(address => DataTypes.UserPosition) storage userPosition,
85      DataTypes.UserConfiguration storage userConfig,
86      DataTypes.ExecuteSwapParams memory params
87  ) external {
88      ExecuteTradeVars memory vars;
89      vars.assetConfigCache[0] = assetConfig[params.shortAsset];
90      vars.assetConfigCache[1] = assetConfig[params.longAsset];
91      executeShorting(
92          reserveConfig[params.shortAsset],
93          vars.assetConfigCache[0],
94          userPosition[params.shortAsset],
95          userConfig,
96          params.amount,
97          true
98      );
99      if (params.applyFees) {
100          vars.feeAmount = params.amount.wadMul(params.tradeFeeMantissa);
101          params.amount = params.amount - vars.feeAmount;
102          IERC20Upgradeable(params.shortAsset).safeTransfer(params.treasury, vars.feeAmount);
103      }
104      if (
105          IERC20Upgradeable(params.shortAsset).allowance(address(this), address(vars.assetConfigCache[0].swapAdapter)) < params.amount
106      ) {
107          IERC20Upgradeable(params.shortAsset).safeApprove(address(vars.assetConfigCache[0].swapAdapter), type(uint256).max);
108      }
109      vars.receivedAmount = vars.assetConfigCache[0].swapAdapter.swap(params.shortAsset, params.longAsset, params.amount, params.data);
110      uint256 increaseShortAmount = GeneralLogic.getAssetUsed(
111          vars.assetConfigCache[0].

```

Figure 6 Source code of *executeTrade* function

Recommendations	It is recommended to add the permission check of the function caller.
Status	Fixed.

```

1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
function swapPosition(address assetIn, address assetOut, uint256 amount, bytes memory data) external onlyLiquidateExecutor {
    TradeLogic.executeTrade(
        _reserveConfig,
        assetConfig,
        _userPosition[LIQUIDATION_WALLET],
        userConfig[LIQUIDATION_WALLET],
        DataTypes.ExecuteSwapParams(
            LIQUIDATION_WALLET,
            treasury,
            assetIn,
            assetOut,
            amount,
            tradeFeeMantissa,
            0,
            0,
            0,
            data,
            false,
            false
        )
    );
}

```

Figure 7 Source code of *swapPosition* function(Fixed)

[LeverFi-3] The *swap* function is improperly designed

Severity Level	Critical
Type	Business Security
Lines	Ledger.sol,ZeroexSwapAdapter.sol#L20-54
Description	When trading, the user can pass in the specified asset for swap, instead of swapping according to the actual asset, so that the user can pass in the fake tokens in exchange for the funds of the protocol.

```

445 function trade(address shortAsset, address longAsset, uint256 amount, bytes memory data) external {
446     _executeTrade(msg.sender, shortAsset, longAsset, amount, data);
447 }
448
449 /**

```

Figure 8 Source code of *trade* function

```

79 /**
80 function executeTrade(
81     mapping(address => DataTypes.ReserveConfig) storage reserveConfig,
82     mapping(address => DataTypes.AssetConfig) storage assetConfig,
83     mapping(address => DataTypes.UserPosition) storage userPosition,
84     DataTypes.UserConfiguration storage userConfig,
85     DataTypes.ExecuteSwapParams memory params
86 ) external {
87     ExecuteTradeVars memory vars;
88     vars.assetConfigCache[0] = assetConfig[params.shortAsset];
89     vars.assetConfigCache[1] = assetConfig[params.longAsset];
90     executeShorting(
91         reserveConfig[params.shortAsset],
92         vars.assetConfigCache[0],
93         userPosition[params.shortAsset],
94         userConfig,
95         params.amount,
96         true
97     );
98     if (params.applyFees) {
99         vars.feeAmount = params.amount.wadMul(params.tradeFeeMantissa);
100         params.amount = params.amount - vars.feeAmount;
101         IERC20Upgradeable(params.shortAsset).safeTransfer(params.treasury, vars.feeAmount);
102     }
103     if (
104         IERC20Upgradeable(params.shortAsset).allowance(address(this), address(vars.assetConfigCache[0].swapAdapter)) < params.amount
105     ) {
106         IERC20Upgradeable(params.shortAsset).safeApprove(address(vars.assetConfigCache[0].swapAdapter), type(uint256).max);
107     }
108     vars.receivedAmount = vars.assetConfigCache[0].swapAdapter.swap(params.shortAsset, params.longAsset, params.amount, params.data);
109     uint256 increaseShortAmount = GeneralLogic.getAssetUsed(
110         vars.assetConfigCache[0].

```

Figure 9 Source code of *executeTrade* function

```

20 /**
21 function swap(address, address, uint256 amountToSwap, bytes memory swapBytesData) external override returns (uint256) {
22     (address approveRouter, address sellingAsset, address buyingAsset, bytes memory swapBytes) = abi.decode(swapBytesData, (address,address,address,bytes));
23     uint256 buyingAssetBalancePrior = IERC20Upgradeable(buyingAsset).balanceOf(address(this));
24     IERC20Upgradeable(sellingAsset).safeTransferFrom(msg.sender, address(this), amountToSwap);
25
26     // approve to appropriate router (Note: router is the part of swapBytes data. Do not take from the swap func arg. The function arg is given because for other swaps like linch
27
28     IERC20Upgradeable(sellingAsset).safeIncreaseAllowance(approveRouter, amountToSwap);
29
30     (bool success, bytes memory data) = approveRouter.call(swapBytes);
31
32     if(!success){
33         assembly {
34             let returndata_size := mload(data)
35             revert(add(32, data), returndata_size)
36         }
37     }
38
39     // check balance after swap
40     uint256 buyingAssetBalance = IERC20Upgradeable(buyingAsset).balanceOf(address(this));
41     uint256 receivedAmount = buyingAssetBalance - buyingAssetBalancePrior;
42
43     IERC20Upgradeable(buyingAsset).safeTransfer(msg.sender, receivedAmount);
44
45     return receivedAmount;
46 }

```

Figure 10 Source code of *swap* function(Unfixed)

Recommendations It is recommended that the code use the incoming params.shortAsset and params.longAsset instead of relying on the selling asset and buy asset in the data entered by the user.

Status Fixed.

```

32 function swap(address sellingAsset, address buyingAsset, uint256 amountToSwap, bytes memory swapBytesData) external override returns (uint256) {
33     (address approveRouter, bytes memory swapBytes) = abi.decode(swapBytesData, (address, address, address, bytes));
34     require(routersList[approveRouter], "disabled router address");
35
36     uint256 buyingAssetBalancePrior = IERC20Upgradeable(buyingAsset).balanceOf(address(this));
37
38     IERC20Upgradeable(sellingAsset).safeTransferFrom(msg.sender, address(this), amountToSwap);
39
40     // approve to appropriate router (Note: router is the part of swapBytes data. Do not take from the swap func arg. The function arg is given because for other swaps like lynch
41     IERC20Upgradeable(sellingAsset).safeIncreaseAllowance(approveRouter, amountToSwap);
42
43     (bool success, bytes memory data) = approveRouter.call(swapBytes);
44
45     if(!success){
46         assembly {
47             let returndata_size := mload(data)
48             revert(add(32, data), returndata_size)
49         }
50     }
51
52     // check balance after swap
53     uint256 buyingAssetBalance = IERC20Upgradeable(buyingAsset).balanceOf(address(this));
54     uint256 receivedAmount = buyingAssetBalance - buyingAssetBalancePrior;
55
56     IERC20Upgradeable(buyingAsset).safeTransfer(msg.sender, receivedAmount);
57     return receivedAmount;
58 }

```

Figure 11 Source code of swap function(Fixed)

[LeverFi-4] The *withdrawLiquidationWalletLong* function is improperly designed

Severity Level	Critical
Type	Business Security
Lines	LiquidationLogic.sol#L277-308
Description	Liquidation wallets can withdraw funds from liquidity providers through this function.

```

276
277
278 function executeWithdrawLiquidationWalletLong(
279     mapping(uint256 => DataTypes.ReserveData) storage reserves,
280     IUserData userData,
281     DataTypes.ExecuteWithdrawLiquidationWalletLong memory params
282 ) external {
283     DataTypes.ReserveData storage reserve = reserves[params.pid];
284
285     reserve.updateIndex();
286
287     DataTypes.ReserveDataCache memory reserveCache = reserve.cache();
288
289     DataTypes.UserPosition memory currPosition = userData.getUserPosition(params.liquidationWallet, params.asset);
290
291     require(currPosition._type == DataTypes.PositionType.Long, Errors.INVALID_POSITION_TYPE);
292
293     if (currPosition.amount < params.amount) {
294         params.amount = currPosition.amount;
295     }
296
297     require(params.amount != 0, Errors.INVALID_ZERO_AMOUNT);
298
299     userData.changePosition(
300         params.liquidationWallet,
301         params.pid,
302         DataTypes.UserPosition(params.amount, DataTypes.PositionType.Short),
303         reserveCache.currBorrowIndexRay,
304         params.decimals
305     );
306
307     IERC20Upgradeable(params.asset).safeTransfer(params.treasury, params.amount);
308
309     emit WithdrawnLiquidationWalletLong(params.asset, params.amount);
310 }
311

```

Figure 12 Source code of *executeWithdrawLiquidationWalletLong* function(Unfixed)

Recommendations	It is recommended that the liquidation wallet can only receive funds through PNL.
Status	Fixed.

```

336 function executeWithdrawLiquidationWalletLong(
337     mapping(address => DataTypes.AssetConfig) storage assetConfigs,
338     mapping(uint256 => DataTypes.ReserveData) storage reserves,
339     mapping(uint256 => DataTypes.CollateralData) storage collaterals,
340     IUserData userData,
341     DataTypes.ExecuteWithdrawLiquidationWalletLong memory params
342 ) external {
343     DataTypes.ReserveData storage reserve = reserves[params.pid];
344
345     reserve.updateIndex();
346
347     DataTypes.ReserveDataCache memory reserveCache = reserve.cache();
348
349     DataTypes.UserPosition memory currPosition = userData.getUserPosition(params.liquidationWallet, params.asset);
350
351     require(currPosition._type == DataTypes.PositionType.Long, Errors.INVALID_POSITION_TYPE);
352
353     if (currPosition.amount < params.amount) {
354         params.amount = currPosition.amount;
355     }
356
357     (,,,, int256 pnlUsd,,,,) = GeneralLogic.getUserLiquidity(
358         reserves,
359         collaterals,
360         assetConfigs,
361         userData,
362         DataTypes.GetUserLiquidityParams(
363             params.liquidationWallet,
364             params.leverageFactor,
365             params.liquidationRatioMantissa
366         )
367     );
368
369     require(pnlUsd > 0, "available leverage not positive");
370
371     (uint256 assetPrice, uint256 assetPriceDecimal) = params.assetConfig.oracle.getAssetPrice(params.asset);
372
373     uint256 assetPriceInWad = assetPrice.unitToWad(assetPriceDecimal);
374
375     uint256 maxAmount = uint256(pnlUsd).wadDiv(assetPriceInWad).wadToUnit(params.assetConfig.decimals);
376
377     if (maxAmount < params.amount) {
378         params.amount = maxAmount;
379     }
380

```

Figure 13 Source code of *executeWithdrawLiquidationWalletLong* function(Fixed)

[LeverFi-5] DoS attack

Severity Level	High
Type	General Vulnerability
Lines	AaveReinvestmentLogic.sol,ConvexReinvestmentLogic.sol#L65,70,110,116-120
Description	<p>There is a DoS vulnerability in the <i>invest</i> function of the AaveReinvestmentLogic and ConvexReinvestmentLogic contracts. If someone sends a token to the contract in advance, the user will fail to call the contract. Because the judgment here is that the amount input by the user must be equal to the contractBalance in the contract, if someone maliciously transfers money to the contract, the judgment will never be satisfied. Similarly, "IERC20Upgradeable(asset).balanceOf(address(this)) == 0" also has this problem.</p>

```

57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
--
function invest(uint256 amount) external override onlyLender {
    IERC20Upgradeable(asset).safeTransferFrom(msg.sender, address(this), amount);
    uint256 contractBalance = IERC20Upgradeable(asset).balanceOf(address(this));
    require(contractBalance > 0, "reinvestment_adapter do not have underlying assets to invest");
    require(contractBalance == amount, "investing amount is different");
    IERC20Upgradeable(asset).safeApprove(platform(), amount);
    IAaveLendingPoolV2(platform()).deposit(asset, amount, address(this), 0);
    require(IERC20Upgradeable(asset).balanceOf(address(this)) == 0, "reinvestment_adapter still has underlying assets to invest");
}

```

Figure 14 Source code of *invest* function (AaveReinvestmentLogic.sol)(Unfixed)

```

101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
--
function invest(
    uint256 amount
) external override onlyLender {
    IERC20Upgradeable(asset).safeTransferFrom(msg.sender, address(this), amount);
    uint256 contractBalance = IERC20Upgradeable(asset).balanceOf(address(this));
    require(contractBalance > 0, "reinvestment_adapter do not have underlying assets to invest");
    require(contractBalance == amount, "investing amount is different");
    IERC20Upgradeable(asset).safeApprove(platform(), amount);
    IConvexBooster(platform()).deposit(poolId(), amount, true);
    require(
        IERC20Upgradeable(asset).balanceOf(address(this)) == 0,
        "reinvestment_adapter still has underlying assets to invest"
    );
}

```

Figure 15 Source code of *invest* function (ConvexReinvestmentLogic.sol)(Unfixed)

Recommendations	It is recommended to delete the code.
Status	Fixed.
	<pre> 58 59 60 61 62 63 64 65 66 67 68 -- function invest(uint256 amount) external override onlyLender { IERC20Upgradeable(asset).safeTransferFrom(msg.sender, address(this), amount); require(IERC20Upgradeable(asset).balanceOf(address(this)) >= amount, "not enough underlying amount to invest"); IERC20Upgradeable(asset).safeApprove(platform(), amount); IAaveLendingPoolV2(platform()).deposit(asset, amount, address(this), 0); } </pre>

Figure 16 Source code of *invest* function (AaveReinvestmentLogic.sol)(Fixed)

```

102  /**
103  * function invest(
104  *   uint256 amount
105  * ) external override onlyLender {
106  *   IERC20Upgradeable(asset).safeTransferFrom(msg.sender, address(this), amount);
107  *
108  *   require(IERC20Upgradeable(asset).balanceOf(address(this)) >= amount, "not enough underlying amount to invest");
109  *
110  *   IERC20Upgradeable(asset).safeApprove(platform(), amount);
111  *   IConvexBooster(platform()).deposit(poolId(), amount, true);
112  * }
113  
```

Figure 17 Source code of *invest* function (ConvexReinvestmentLogic.sol)(Fixed)

[LeverFi-6] The `_calculateRewards` function design flaw

Severity Level	High
Type	Business Security
Lines	ConvexReinvestmentLogic.sol#L189-226
Description	Design flaw in the <code>_calculateRewards</code> function of the <code>ConvexReinvestmentLogic</code> contract. Because the user does not update the value of <code>acquiredBalance</code> after receiving the reward, it will result in an error in the calculation of the reward.

```

188
189
190     function _calculateRewards(
191         address user,
192         uint256 currBalance,
193         RewardConfig memory reward,
194         UserReward memory userReward,
195         bool isClaim
196     ) internal returns (RewardConfig memory, UserReward memory) {
197
198         address crvRewards = IConvexBooster(platform()).poolInfo(poolId()).crvRewards;
199         IConvexRewards(crvRewards).getReward();
200         uint256 accruedBalance = IERC20Upgradeable(reward.asset).balanceOf(address(this));
201
202         if (totalSupply() > 0 && accruedBalance > reward.remaining) {
203             reward.integral += (accruedBalance - reward.remaining) * 1e20 / totalSupply();
204         }
205
206         if (isClaim || reward.integral > userReward.integral) {
207             uint256 receivable = (reward.integral - userReward.integral) * currBalance / 1e20;
208
209             if (isClaim) {
210                 receivable += userReward.claimable;
211
212                 IERC20Upgradeable(reward.asset).safeTransfer(user, receivable);
213                 userReward.claimable = 0;
214             } else {
215                 userReward.claimable += receivable;
216             }
217             userReward.integral = reward.integral;
218
219             if (accruedBalance != reward.remaining) {
220                 reward.remaining = accruedBalance;
221             }
222
223             return (reward, userReward);
224         }
225
226         return (reward, userReward);
227     }
228

```

Figure 18 Source code of `_calculateRewards` function(Unfixed)

Recommendations	It is recommended to update the <code>acquiredBalance</code> after claiming the reward.
Status	Fixed.

```

180
181 function _calculateRewards(
182     address user,
183     uint256 currBalance,
184     RewardConfig memory reward,
185     UserReward memory userReward,
186     bool isClaim
187 ) internal returns (RewardConfig memory, UserReward memory) {
188
189     address crvRewards = IConvexBooster(platform()).poolInfo(poolId()).crvRewards;
190     IConvexRewards(crvRewards).getReward();
191
192     uint256 accruedBalance = IERC20Upgradeable(reward.asset).balanceOf(address(this));
193
194     if (totalSupply() > 0 && accruedBalance > reward.remaining) {
195         reward.integral += (accruedBalance - reward.remaining) * 1e20 / totalSupply();
196     }
197
198     if (isClaim || reward.integral > userReward.integral) {
199         uint256 receivable = (reward.integral - userReward.integral) * currBalance / 1e20;
200
201         if (isClaim) {
202             receivable += userReward.claimable;
203
204             IERC20Upgradeable(reward.asset).safeTransfer(user, receivable);
205             userReward.claimable = 0;
206             accruedBalance -= receivable;
207         } else {
208             userReward.claimable += receivable;
209         }
210         userReward.integral = reward.integral;
211     }
212
213     if (accruedBalance != reward.remaining) {
214         reward.remaining = accruedBalance;
215     }
216
217     return (reward, userReward);
218 }
219

```

Figure 19 Source code of `_calculateRewards` function (ConvexReinvestmentLogic.sol)(Fixed)

[LeverFi-7] The `_depositReserve` function is improperly designed

Severity Level	High
Type	Business Security
Lines	Ledger.sol#L812-825
Description	<p>First of all, in the <code>getReserveSupply</code> function, if the function uses true, then <code>poolAmount</code> will not use the <code>UtilizedSupply</code> part. Go back to the <code>_depositReserve</code> function of the Ledger contract and calculate the part of <code>shareAmountRay</code> that does not use <code>UtilizedSupply</code>, then the calculated share will be wrong. In the <code>_withdrawReserve</code> function, the <code>poolAmount</code> is added with <code>UtilizedSupply</code>. The calculated <code>shareAmountRay</code> is smaller than expected, which will lead to arbitrage space. Similarly, the part of <code>UtilizedSupply</code> should also be used when calculating <code>currUserReserveBalance</code>.</p>

```

798 //***** INTERNAL *****/
799 function _depositReserve(address user, address asset, uint256 amount) internal {
800     DataTypes.AssetConfig memory assetConfigCache = assetConfig[asset];
801     DataTypes.ReserveConfig storage reserve = _reserveConfig[asset];
802     uint256 unit = assetConfigCache.decimals;
803
804     ValidationLogic.validateDepositReserve(reserve, assetConfigCache, amount);
805
806     reserve.updateIndex();
807
808     uint256 currUserReserveBalance;
809     // is first deposit
810     if (userReserveShareRay[user][asset] == 0) {
811         userConfig[user].setUsingReserve(reserve.assetId, true);
812     } else {
813         currUserReserveBalance = ShareBaseAccounting.getAbsoluteAmount(
814             userReserveShareRay[user][asset],
815             reserve.scaledTotalSupplyRay,
816             reserve.getReserveSupply(assetConfigCache, true).unitToRay(unit)
817             ).rayToUnit(unit);
818     }
819
820     uint256 shareAmountRay = ShareBaseAccounting.getShareAmount(
821         amount.unitToRay(unit),
822         reserve.scaledTotalSupplyRay,
823         reserve.getReserveSupply(assetConfigCache, true).unitToRay(unit)
824     );
825
826     // update user info
827     userReserveShareRay[user][asset] += shareAmountRay;
828     reserve.scaledTotalSupplyRay += shareAmountRay;
829
830     IERC20Upgradeable(asset).safeTransferFrom(user, address(this), amount);
831
832     if (reserve.reinvestment != address(0)) {
833         if (IERC20Upgradeable(reserve.asset).allowance(address(this), reserve.reinvestment) < amount) {
834             IERC20Upgradeable(reserve.asset).safeApprove(reserve.reinvestment, type(uint256).max);
835         }
836
837         IReinvestment(reserve.reinvestment).checkpoint(user, currUserReserveBalance);
838         IReinvestment(reserve.reinvestment).invest(amount);
839     } else {
840         reserve.liquidSupply += amount;
841     }
842
843     emit DepositedReserve(user, asset, reserve.reinvestment, amount);
844 }
845

```

Figure 20 Source code of `_depositReserve` function(Unfixed)

Recommendations	It is recommended to set the <code>getReserveSupply</code> function to false.
Status	Fixed.


```

844 ✓ function _depositReserve(address user, address asset, uint256 amount) internal {
845     DataTypes.AssetConfig memory assetConfigCache = assetConfig[asset];
846     DataTypes.ReserveConfig storage reserve = _reserveConfig[asset];
847     uint256 unit = assetConfigCache.decimals;
848
849     ValidationLogic.validateDepositReserve(reserve, assetConfigCache, amount);
850
851     reserve.updateIndex();
852     uint256 currReserveSupply = reserve.getReserveSupply(assetConfigCache, false);
853     uint256 currUserReserveBalance;
854
855     // is first deposit
856 ✓ if (userReserveShareRay[user][asset] == 0) {
857     userConfig[user].setUsingReserve(reserve.assetId, true);
858 ✓ } else {
859 ✓     currUserReserveBalance = ShareBaseAccounting.getAbsoluteAmount(
860         userReserveShareRay[user][asset],
861         reserve.scaledTotalSupplyRay,
862         currReserveSupply.unitToRay(unit)
863     ).rayToUnit(unit);
864 }
865
866 ✓ uint256 shareAmountRay = ShareBaseAccounting.getShareAmount(
867     amount.unitToRay(unit),
868     reserve.scaledTotalSupplyRay,
869     currReserveSupply.unitToRay(unit)
870 );
871
872 // update user info
873 userReserveShareRay[user][asset] += shareAmountRay;
874 reserve.scaledTotalSupplyRay += shareAmountRay;
875
876 IERC20Upgradeable(asset).safeTransferFrom(user, address(this), amount);
877
878 ✓ if (reserve.reinvestment != address(0)) {
879 ✓     if (IERC20Upgradeable(reserve.asset).allowance(address(this), reserve.reinvestment) < amount) {
880         IERC20Upgradeable(reserve.asset).safeApprove(reserve.reinvestment, 0);
881         IERC20Upgradeable(reserve.asset).safeApprove(reserve.reinvestment, type(uint256).max);
882     }
883
884     IReinvestment(reserve.reinvestment).checkpoint(user, currUserReserveBalance);
885     IReinvestment(reserve.reinvestment).invest(amount);
886 ✓ } else {
887     reserve.liquidSupply += amount;
888 }
889
890 emit DepositedReserve(user, asset, reserve.reinvestment, amount);
891 }

```

Figure 21 Source code of `_depositReserve` function(Fixed)

[LeverFi-8] The *configureAsset* function is improperly designed

Severity Level	High
Type	Business Security
Lines	Ledger.sol#L231
Description	In the Ledger contract, when the <i>configureAsset</i> function modifies the configuration, it will replace <i>assetLength</i> with the latest index, which will cause data confusion.

```

220  */
221  function configureAsset(
222      address asset,
223      uint256 decimals,
224      address longReinvestment,
225      ISwapAdapter swapAdapter,
226      IPriceOracleGetter oracle,
227      DataTypes.AssetState state,
228      DataTypes.AssetMode mode
229  ) public onlyOperator {
230      assetConfig[asset] = DataTypes.AssetConfig(
231          assetLength,
232          decimals,
233          longReinvestment,
234          swapAdapter,
235          oracle,
236          state,
237          mode
238      );
239
240      emit ConfiguredAsset(asset, decimals, longReinvestment, address(swapAdapter), address(oracle), state, mode);
241  }
242  ...

```

Figure 22 Source code of *configureAsset* function(Unfixed)

Recommendations	It is recommended that <i>assetId</i> should remain unchanged when set.
-----------------	---

Status	Fixed.
--------	--------

```

234  function configureAsset(
235      address asset,
236      uint256 decimals,
237      address longReinvestment,
238      ISwapAdapter swapAdapter,
239      IPriceOracleGetter oracle,
240      DataTypes.AssetState state,
241      DataTypes.AssetMode mode
242  ) public onlyOperator {
243      DataTypes.AssetConfig storage _assetConfig = assetConfig[asset];
244
245      _assetConfig.decimals = decimals;
246      _assetConfig.longReinvestment = longReinvestment;
247      _assetConfig.swapAdapter = swapAdapter;
248      _assetConfig.oracle = oracle;
249      _assetConfig.state = state;
250      _assetConfig.mode = mode;
251
252      emit ConfiguredAsset(asset, decimals, longReinvestment, address(swapAdapter), address(oracle), state, mode);
253  }
254

```

Figure 23 Source code of *configureAsset* function(Fixed)

[LeverFi-9] Clearing mechanism error

Severity Level	High
Type	Business Security
Lines	LiquidationLogic.sol#L189-221
Description	When liquidation is performed in the <i>executeForeclosure</i> function, the debt repaid by the liquidation wallet is not recorded correctly, which will cause the liquidation wallet to use a small amount of funds to pay off a large debt.

```

188     } else {
189         uint256 userShortAbsoluteAmount = userAssetPosition.amount
190         .rayMul(vars.reserveConfigCache.currBorrowIndexRay)
191         .rayToUnit(vars.assetConfig.decimals);
192
193         uint256 amountToRepay = userShortAbsoluteAmount;
194         // repay user short with liquidation wallet if has any
195         if (liquidationAssetPosition.amount > 0 && liquidationAssetPosition._type == DataTypes.PositionType.Long) {
196             if (userShortAbsoluteAmount >= liquidationAssetPosition.amount) {
197                 amountToRepay = liquidationAssetPosition.amount;
198             }
199             TradeLogic.executeLonging(
200                 reserveConfigs[vars.asset],
201                 vars.assetConfig,
202                 userAssetPosition,
203                 userConfigs[vars.user],
204                 params.treasury,
205                 amountToRepay,
206                 false
207             );
208         }
209         vars.newPosition = TradeLogic.getNewPosition(
210             vars.assetConfig,
211             liquidationAssetPosition,
212             DataTypes.UserPosition[amountToRepay], DataTypes.PositionType.Short,
213             vars.reserveConfigCache.currBorrowIndexRay
214         );
215         liquidationAssetPosition.amount = vars.newPosition.amount;
216         liquidationAssetPosition._type = vars.newPosition._type;
217         userConfigs[params.liquidationWallet].setUsingPosition(vars.poolId, vars.newPosition.amount > 0);
218         userAssetPosition.amount = 0;
219         userAssetPosition._type = DataTypes.PositionType.Long;
220         userConfigs[vars.user].setUsingPosition(vars.poolId, false);
221     }
222 }

```

Figure 24 Source code of *executeForeclosure* function(Unfixed)

Recommendations It is recommended to use the actual amount, not the amount paid for the user.

Status Fixed.

```

191     } else {
192         uint256 userShortAbsoluteAmount = userAssetPosition.amount
193         .rayMul(vars.reserveConfigCache.currBorrowIndexRay)
194         .rayToUnit(vars.assetConfig.decimals);
195
196         uint256 amountToRepay = userShortAbsoluteAmount;
197         // repay user short with liquidation wallet if has any
198         if (liquidationAssetPosition.amount > 0 && liquidationAssetPosition._type == DataTypes.PositionType.Long) {
199             if (userShortAbsoluteAmount >= liquidationAssetPosition.amount) {
200                 amountToRepay = liquidationAssetPosition.amount;
201             }
202             TradeLogic.executeLonging(
203                 reserveConfigs[vars.asset],
204                 vars.assetConfig,
205                 userAssetPosition,
206                 userConfigs[vars.user],
207                 params.treasury,
208                 amountToRepay,
209                 false
210             );
211         }
212         vars.newPosition = TradeLogic.getNewPosition(
213             vars.assetConfig,
214             liquidationAssetPosition,
215             DataTypes.UserPosition[userShortAbsoluteAmount], DataTypes.PositionType.Short,
216             vars.reserveConfigCache.currBorrowIndexRay
217         );
218         liquidationAssetPosition.amount = vars.newPosition.amount;
219         liquidationAssetPosition._type = vars.newPosition._type;
220         userConfigs[params.liquidationWallet].setUsingPosition(vars.poolId, vars.newPosition.amount > 0);
221         userAssetPosition.amount = 0;
222         userAssetPosition._type = DataTypes.PositionType.Long;
223         userConfigs[vars.user].setUsingPosition(vars.poolId, false);
224     }
225 }

```

Figure 25 Source code of *executeForeclosure* function(Fixed)

[LeverFi-10] The *getNormalizedDebt* function is improperly designed

Severity Level	Medium
Type	Business Security
Lines	ReserveLogic.sol#L59-64
Description	<p>When querying currBorrowIndexRay through the <i>getNormalizedDebt</i> function, if user B borrows money right after user A at the same time. At this time, according to the code logic, their currBorrowIndexRay is the same, but when user B borrows money, the utilization rate changes. Theoretically currBorrowIndexRay will be different, which will result in less interest repaid by users who borrow later.</p>

```

55     function getNormalizedDebt(
56         DataTypes.ReserveConfig memory reserve
57     ) internal view returns (uint256, uint256, uint256) {
58         uint256 timestamp = reserve.lastUpdatedTimestamp;
59         if (timestamp == block.timestamp) {
60             return (
61                 reserve.reserveIndexRay,
62                 reserve.protocolIndexRay,
63                 reserve.reserveIndexRay + reserve.protocolIndexRay
64             );
65         } else {
66             uint256 currBorrowIndexRay = reserve.reserveIndexRay + reserve.protocolIndexRay;
67             uint256 interestRateRay = getInterestRate(
68                 reserve.scaledUtilizedSupplyRay,
69                 reserve.scaledTotalSupplyRay,
70                 reserve.protocolRateMantissaRay,
71                 reserve.utilizationBaseRateMantissaRay,
72                 reserve.kinkMantissaRay,
73                 reserve.multiplierAnnualRay,
74                 reserve.jumpMultiplierAnnualRay
75             );
76             if (interestRateRay == 0) {
77                 return (
78                     reserve.reserveIndexRay,
79                     reserve.protocolIndexRay,
80                     reserve.reserveIndexRay + reserve.protocolIndexRay
81                 );
82             }

```

Figure 26 Source code of *getNormalizedDebt* function(Unfixed)

Recommendations	It is recommended to delete the if judgment.
Status	Fixed.

```

59     function getNormalizedDebt(
60         DataTypes.ReserveConfig memory reserve
61     ) internal view returns (uint256, uint256, uint256) {
62         uint256 currBorrowIndexRay = reserve.reserveIndexRay + reserve.protocolIndexRay;
63         uint256 interestRateRay = getInterestRate(
64             reserve.scaledUtilizedSupplyRay,
65             reserve.scaledTotalSupplyRay,
66             reserve.protocolRateMantissaRay,
67             reserve.utilizationBaseRateMantissaRay,
68             reserve.kinkMantissaRay,
69             reserve.multiplierAnnualRay,
70             reserve.jumpMultiplierAnnualRay
71         );
72         if (interestRateRay == 0) {
73             return (
74                 reserve.reserveIndexRay,
75                 reserve.protocolIndexRay,
76                 reserve.reserveIndexRay + reserve.protocolIndexRay
77             );
78         }
79         uint256 cumulatedInterestIndexRay = InterestUtils.getCompoundedInterest(
80             interestRateRay, reserve.lastUpdatedTimestamp, block.timestamp
81         );
82         uint256 growthIndexRay = currBorrowIndexRay.rayMul(cumulatedInterestIndexRay) - currBorrowIndexRay;
83         uint256 protocolInterestRatio = reserve.protocolRateMantissaRay.rayDiv(interestRateRay);
84         uint256 nextReserveIndexRay = reserve.reserveIndexRay + growthIndexRay.rayMul(MathUtils.RAY - protocolInterestRatio);
85         uint256 nextProtocolIndexRay = reserve.protocolIndexRay + growthIndexRay.rayMul(protocolInterestRatio);
86         return (
87             nextReserveIndexRay,
88             nextProtocolIndexRay,
89             nextProtocolIndexRay + nextReserveIndexRay
90         );
91     }
92 }

```

Figure 27 Source code of *getNormalizedDebt* function(Fixed)

[LeverFi-11] Implementation flaws in the *reinvestReserveSupply* and *reinvestCollateralSupply* functions

Severity Level	Medium
Type	Business Security
Lines	Ledger.sol#L759-795
Description	In the <i>reinvestReserveSupply</i> and <i>reinvestCollateralSupply</i> functions, the lack of approve for New_reinvestment will cause the function call to fail.

```

759  /**
760  function reinvestReserveSupply(address asset) external onlyOperator {
761
762      DataTypes.ReserveConfig storage reserveConfig = _reserveConfig[asset];
763
764      require(reserveConfig.state == DataTypes.AssetState.Disabled, "state is not disabled");
765      require(reserveConfig.reinvestment != address(0), "reinvestment is not set");
766
767      IReinvestment(reserveConfig.reinvestment).invest(reserveConfig.liquidSupply);
768
769      emit ReinvestedReserveSupply(asset, reserveConfig.liquidSupply);
770
771      reserveConfig.liquidSupply = 0;
772  }
773
774  /**
775  * @notice Reinvests collateral supply
776  * @param asset Underlying asset address
777  * @param reinvestment Address where the asset is reinvested in
778  */
779  function reinvestCollateralSupply(address asset, address reinvestment) external onlyOperator {
780
781      uint256 poolId = collateralPoolList[asset][reinvestment];
782      DataTypes.CollateralConfig storage collateralConfig = _collateralConfig[poolId];
783
784      require(collateralConfig.state == DataTypes.AssetState.Disabled, "state is not disabled");
785      require(collateralConfig.reinvestment != address(0), "reinvestment is not set");
786
787      IReinvestment(collateralConfig.reinvestment).invest(collateralConfig.liquidSupply);
788
789      emit ReinvestedCollateralSupply(asset, reinvestment, collateralConfig.liquidSupply);
790
791      collateralConfig.liquidSupply = 0;
792  }
793
794
795
796

```

Figure 28 Source code of related functions(Unfixed)

Recommendations	It is recommended to increase approve.
Status	Fixed.

```

802  */
803  function reinvestReserveSupply(address asset) external onlyOperator {
804
805      DataTypes.ReserveConfig storage reserveConfig = _reserveConfig[asset];
806
807      require(reserveConfig.state == DataTypes.AssetState.Disabled, "state is not disabled");
808      require(reserveConfig.reinvestment != address(0), "reinvestment is not set");
809
810      IERC20Upgradeable(asset).safeApprove(reserveConfig.reinvestment, reserveConfig.liquidSupply);
811      IReinvestment(reserveConfig.reinvestment).invest(reserveConfig.liquidSupply);
812
813      emit ReinvestedReserveSupply(asset, reserveConfig.liquidSupply);
814
815      reserveConfig.liquidSupply = 0;
816
817  }
818
819  /**
820   * @notice Reinvests collateral supply
821   * @param asset Underlying asset address
822   * @param reinvestment Address where the asset is reinvested in
823   */
824  function reinvestCollateralSupply(address asset, address reinvestment) external onlyOperator {
825
826      uint256 poolId = collateralPoolList[asset][reinvestment];
827      DataTypes.CollateralConfig storage collateralConfig = _collateralConfig[poolId];
828
829      require(collateralConfig.state == DataTypes.AssetState.Disabled, "state is not disabled");
830      require(collateralConfig.reinvestment != address(0), "reinvestment is not set");
831
832      IERC20Upgradeable(asset).safeApprove(collateralConfig.reinvestment, collateralConfig.liquidSupply);
833      IReinvestment(collateralConfig.reinvestment).invest(collateralConfig.liquidSupply);
834
835      emit ReinvestedCollateralSupply(asset, reinvestment, collateralConfig.liquidSupply);
836
837      collateralConfig.liquidSupply = 0;
838
839  }
840
841

```

Figure 29 Source code of related functions(Fixed)

[LeverFi-12] Unsafe call

Severity Level	Medium
Type	General Vulnerability
Lines	ZeroexSwapAdapter.sol#L35
Description	<p>Unsafe call method. In the <i>swap</i> function of the ZeroexSwapAdapter contract, because the swapBytesData parameter is controllable, the approveRouter and swapBytes are also controllable. Then an attacker can use call to perform arbitrary operations. For example, approveRouter is specified as a token, and the swapBytes parameter is the token authorization behavior (the contract is authorized to the attacker), then the attacker can withdraw the tokens in the contract.</p> <pre> 17 18 19 20 function swap(address, address, uint256 amountToSwap, bytes memory swapBytesData) external override returns (uint256) { 21 22 (address approveRouter, address sellingAsset, address buyingAsset, bytes memory swapBytes) = abi.decode(swapBytesData, (address,address,address,bytes)); 23 24 uint256 buyingAssetBalancePrior = IERC20Upgradeable(buyingAsset).balanceOf(address(this)); 25 26 IERC20Upgradeable(sellingAsset).safeTransferFrom(msg.sender, address(this), amountToSwap); 27 28 29 30 31 // approve to appropriate router (Note: router is the part of swapBytes data. Do not take from the swap func arg. The function arg is given because for other swaps like 1inch 32 33 IERC20Upgradeable(sellingAsset).safeIncreaseAllowance(approveRouter, amountToSwap); 34 35 (bool success, bytes memory data) = approveRouter.call(swapBytes); 36 37 if(!success){ 38 assembly { 39 let returndata_size := mload(data) 40 revert(add(32, data), returndata_size) 41 } 42 } 43 44 // check balance after swap 45 uint256 buyingAssetBalance = IERC20Upgradeable(buyingAsset).balanceOf(address(this)); 46 uint256 receivedAmount = buyingAssetBalance - buyingAssetBalancePrior; 47 48 49 50 IERC20Upgradeable(buyingAsset).safeTransfer(msg.sender, receivedAmount); 51 52 return receivedAmount; 53 54 } 55 </pre>

Figure 30 Source code of *swap* function (ZeroexSwapAdapter.sol)(Unfixed)

Recommendations	It is recommended to directly call this interface for swap instead of using the call method.
-----------------	--

Status	Fixed.
	<pre> 20 21 // @param router Router address to enable 22 function configureRoutersList(address router, bool value) external onlyOwner { 23 emit ConfiguredRoutersList(router, routersList[router], value); 24 routersList[router] = value; 25 } </pre>

Figure 31 Source code of *configureRoutersList* function (ZeroexSwapAdapter.sol)(Fixed)

[LeverFi-13] The *configureReserve* function design flaw

Severity Level	Medium
Type	Business Security
Lines	Ledger.sol#L304
Description	<p>First, in the <i>configureReserve</i> function, the operator can modify the reinvestment arbitrarily. It is assumed that the operator initially sets the reinvestment to a non-zero address, and the user stakes during this period, and then the operator sets the reinvestment to zero. At this time, when the user stakes in, the calculation of the <i>getShareAmount</i> function in the <i>_depositReserve</i> function will be wrong, because the <i>getReserveAvailableSupply</i> function here can only obtain the total amount of <i>reserve.reinvestment</i> that is zero, and does not obtain the amount of reinvestment that is not zero, so the calculation error is caused.</p>

```

290 function configureReserve(
291     address asset,
292     address reinvestment,
293     uint256 feeMantissaRay,
294     uint256 protocolRateMantissaRay,
295     uint256 utilizationBaseRateMantissaRay,
296     uint256 kinkMantissaRay,
297     uint256 multiplierPerAnnualRay,
298     uint256 jumpMultiplierPerAnnualRay,
299     DataTypes.AssetState state
300 ) public onlyOperator {
301     DataTypes.ReserveConfig storage reserve = _reserveConfig[asset];
302
303     require(jumpMultiplierPerAnnualRay >= multiplierPerAnnualRay, "Jump multiplier must be greater or equal normal multiplier");
304     reserve.reinvestment = reinvestment;
305     reserve.feeMantissaRay = feeMantissaRay;
306     reserve.protocolRateMantissaRay = protocolRateMantissaRay;
307     reserve.utilizationBaseRateMantissaRay = utilizationBaseRateMantissaRay;
308     reserve.kinkMantissaRay = kinkMantissaRay;
309     reserve.multiplierAnnualRay = multiplierPerAnnualRay;
310     reserve.jumpMultiplierAnnualRay = jumpMultiplierPerAnnualRay;
311     reserve.state = state;
312
313     emit ConfiguredReserve(
314         asset,
315         reinvestment,
316         feeMantissaRay,
317         utilizationBaseRateMantissaRay,
318         kinkMantissaRay,
319         multiplierPerAnnualRay,
320         jumpMultiplierPerAnnualRay,
321         state
322     );
323 }

```

Figure 32 Source code of *configureReserve* function (Ledger.sol)(Unfixed)

```

797 /***** INTERNAL *****/
798 function _depositReserve(address user, address asset, uint256 amount) internal {
799     DataTypes.AssetConfig memory assetConfigCache = assetConfig[asset];
800     DataTypes.ReserveConfig storage reserve = _reserveConfig[asset];
801     uint256 unit = assetConfigCache.decimals;
802
803     ValidationLogic.validateDepositReserve(reserve, assetConfigCache, amount);
804
805     reserve.updateIndex();
806
807     uint256 currUserReserveBalance;
808     // is first deposit
809     if (userReserveShareRay[user][asset] == 0) {
810         userConfig[user].setUsingReserve(reserve.assetId, true);
811     } else {
812         currUserReserveBalance = ShareBaseAccounting.getAbsoluteAmount(
813             userReserveShareRay[user][asset],
814             reserve.scaledTotalSupplyRay,
815             reserve.getReserveSupply(assetConfigCache, true).unitToRay(unit)
816         ).rayToUnit(unit);
817     }
818
819     uint256 shareAmountRay = ShareBaseAccounting.getShareAmount(
820         amount.unitToRay(unit),
821         reserve.scaledTotalSupplyRay,
822         reserve.getReserveSupply(assetConfigCache, true).unitToRay(unit)
823     );
824

```

Figure 33 Source code of `_depositReserve` function (Ledger.sol)(Unfixed)

```

22 function getReserveSupply(
23     DataTypes.ReserveConfig storage reserve,
24     DataTypes.AssetConfig memory assetConfig,
25     bool claimable
26 ) internal view returns (uint256 poolAmount) {
27     if (!claimable) {
28         poolAmount += getUtilizedSupply(reserve, assetConfig);
29     }
30     poolAmount += GeneralLogic.getReserveAvailableSupply(reserve);
31 }
32

```

Figure 34 Source code of `getReserveSupply` function (ReserveLogic.sol)(Unfixed)

```

25 function getReserveAvailableSupply(
26     DataTypes.ReserveConfig memory reserve
27 ) public view returns (uint256 supply) {
28     if (reserve.reinvestment == address(0)) {
29         supply = reserve.liquidSupply;
30     } else {
31         supply = IReinvestment(reserve.reinvestment).totalSupply();
32     }
33 }

```

Figure 35 Source code of `getReserveAvailableSupply` function (GeneralLogic.sol)(Unfixed)

Recommendations It is recommended that the project side add a state in which users cannot stake.

Status Fixed. The project party has added this function to avoid the situation that users can still stake when modifying the reinvestment.

```

338     */
339     function configureReserveReinvestment(
340         address asset,
341         address reinvestment
342     ) public onlyOperator {
343         DataTypes.ReserveConfig storage reserve = _reserveConfig[asset];
344
345         require(reserve.assetId != 0, "asset reserve is not initialized");
346         require(reserve.state == DataTypes.AssetState.Disabled);
347
348         emit ConfiguredReserveReinvestment(asset, reserve.reinvestment, reinvestment);
349
350         reserve.reinvestment = reinvestment;
351     }
352

```

Figure 36 Source code of *configureReserveReinvestment* function (Ledger.sol)(Fixed)

[LeverFi-14] Centralization risk

Severity Level	Medium
Type	Business Security
Lines	Ledger.sol
Description	In the Ledger contract, the operator permission can modify the relevant parameters in the contract, which is high permission.

```

140 ~/
147 function setLeverageFactor(uint256 leverageFactor_) external onlyOperator {
148     require(leverageFactor_ >= 1e18, "leverage factor cannot be lower than 1x");
149     require(leverageFactor_ <= 10e18, "leverage factor cannot be lower than 10x");
150
151     emit LeverageFactorUpdated(leverageFactor, leverageFactor_);
152
153     leverageFactor = leverageFactor_;
154 }
155
156 /**
157  * @notice Setter for tradeFeeMantissa
158  * @param tradeFeeMantissa_ The new tradeFeeMantissa
159  */
160 function setTradeFee(uint256 tradeFeeMantissa_) external onlyOperator {
161     require(tradeFeeMantissa_ <= 0.1e18, "fee cannot be more than 10%");
162
163     emit TradeFeeUpdated(tradeFeeMantissa, tradeFeeMantissa_);
164
165     tradeFeeMantissa = tradeFeeMantissa_;
166 }
167
168 /**
169  * @notice Setter for liquidationRatioMantissa
170  * @param liquidationRatioMantissa_ The new liquidationRatioMantissa
171  */
172 function setLiquidationRatio(uint256 liquidationRatioMantissa_) external onlyOperator {
173     require(liquidationRatioMantissa_ >= 0.5e18, "fee cannot be less than 50%");
174     require(liquidationRatioMantissa_ <= 0.9e18, "fee cannot be more than 90%");
175
176     emit LiquidationRatioUpdated(liquidationRatioMantissa, liquidationRatioMantissa_);
177
178     liquidationRatioMantissa = liquidationRatioMantissa_;
179 }
180
181 /***** CORE FUNCTIONS *****/

```

Figure 37 Source code of related functions

```

234 function configureAsset(
235     address asset,
236     uint256 decimals,
237     address longReinvestment,
238     ISwapAdapter swapAdapter,
239     IPriceOracleGetter oracle,
240     DataTypes.AssetState state,
241     DataTypes.AssetMode mode
242 ) public onlyOperator {
243     DataTypes.AssetConfig storage _assetConfig = assetConfig[asset];
244
245     _assetConfig.decimals = decimals;
246     _assetConfig.longReinvestment = longReinvestment;
247     _assetConfig.swapAdapter = swapAdapter;
248     _assetConfig.oracle = oracle;
249     _assetConfig.state = state;
250     _assetConfig.mode = mode;
251
252     emit ConfiguredAsset(asset, decimals, longReinvestment, address(swapAdapter), address(oracle), state, mode);
253 }
254

```

Figure 38 Source code of *configureAsset* function

Recommendations	It is recommended to use a multi-signature wallet to manage operator permissions.
Status	Fixed. The project party is planned that a Gnosis multi-signature wallet will be used as

operator for the Ledger.

[LeverFi-15] The `_repayShort` function is improperly designed

Severity Level	Medium
Type	Business Security
Lines	Ledger.sol#1074-1107
Description	The essence of the <code>_repayShort</code> function is that the user repays the debt, but the user can use the <code>_repayShort</code> function to add positions. Because the function does not determine whether the user's position is long, and the money transferred by the user is not transferred to the investment contract. When the last user wants to withdraw funds, it will cause the call to fail because the user withdraws funds from the investment contract.

```

1074 function _repayShort(address user, address asset, uint256 amount) internal {
1075     _reserveConfig[asset].updateIndex();
1076
1077     (uint256 positionAbsoluteAmount,) = GeneralLogic.getUserPosition(
1078         assetConfig[asset],
1079         _userPosition[user][asset],
1080         _reserveConfig[asset].getBorrowIndex()
1081     );
1082
1083     // cap amount to max repayable
1084     if (amount > positionAbsoluteAmount) {
1085         amount = positionAbsoluteAmount;
1086     }
1087
1088     ValidationLogic.validateRepayShort(
1089         userLastTradeBlock[user],
1090         DataTypes.ValidateRepayShortParams(
1091             user,
1092             asset,
1093             amount
1094         )
1095     );
1096
1097     IERC20Upgradeable(asset).safeTransferFrom(user, address(this), amount);
1098
1099     TradeLogic.executeLonging(
1100         _reserveConfig[asset],
1101         assetConfig[asset],
1102         _userPosition[user][asset],
1103         userConfig[user],
1104         treasury,
1105         amount,
1106         false
1107     );
1108

```

Figure 39 Source code of `_repayShort` function(Unfixed)

Recommendations	It is recommended to judge whether the user's position is a short type in <code>validateRepayShort</code> .
Status	Fixed.

```

1074 function _repayShort(address user, address asset, uint256 amount) internal {
1075     _reserveConfig[asset].updateIndex();
1076
1077     (uint256 positionAbsoluteAmount,) = GeneralLogic.getUserPosition(
1078         assetConfig[asset],
1079         _userPosition[user][asset],
1080         _reserveConfig[asset].getBorrowIndex()
1081     );
1082
1083     // cap amount to max repayable
1084     if (amount > positionAbsoluteAmount) {
1085         amount = positionAbsoluteAmount;
1086     }
1087
1088     ValidationLogic.validateRepayShort(
1089         userLastTradeBlock[user],
1090         DataTypes.ValidateRepayShortParams(
1091             user,
1092             asset,
1093             amount
1094         )
1095     );
1096
1097     IERC20Upgradeable(asset).safeTransferFrom(user, address(this), amount);
1098
1099     TradeLogic.executeLonging(
1100         _reserveConfig[asset],
1101         assetConfig[asset],
1102         _userPosition[user][asset],
1103         userConfig[user],
1104         treasury,
1105         amount,
1106         true
1107     );
1108
1109     emit RepaidShort(user, asset, amount);
  
```

Figure 40 Source code of `_repayShort` function(Fixed)

[LeverFi-16] The *ExecutewithdrawReserve* function lacks update percentageray parameters

Severity Level	Medium
Type	Business Security
Lines	ReservePoolLogic.sol#96-144
Description	The <i>UtilizationPercentageRay</i> parameter is not updated in the <i>ExecutewithDrawReserve</i> function, which will result in inaccurate utilization if the user exits liquidity.

```

129  if (localReserve.configuration.depositFeeMantissaGwei > 0) {
130      withdrawalFee = amount.wadMul(
131          uint256(localReserve.configuration.depositFeeMantissaGwei).unitToWad(9)
132      );
133
134      IERC20Upgradeable(asset).safeTransfer(protocolConfig.treasury, withdrawalFee);
135  }
136
137  if (localReserve.ext.bonusPool != address(0)) {
138      uint256 nextUserReserveBalance = IUserData(protocolConfig.userData).getUserReserve(user, asset, false);
139      IBonusPool(localReserve.ext.bonusPool).updatePoolUser(asset, user, nextUserReserveBalance);
140  }
141
142  IERC20Upgradeable(asset).safeTransfer(user, amount - withdrawalFee);
143
144  emit WithdrawnReserve(user, asset, localReserve.ext.reinvestment, amount - withdrawalFee);
145  }
146

```

Figure 41 Source code of *executeWithdrawReserve* function(Unfixed)

Recommendations	It is recommended to increase "reserve.postupDateRESERVADATA();" to update the utilization rate.
-----------------	--

Status	Fixed.
--------	--------

```

128  uint256 withdrawalFee;
129  if (localReserve.configuration.depositFeeMantissaGwei > 0) {
130      withdrawalFee = amount.wadMul(
131          uint256(localReserve.configuration.depositFeeMantissaGwei).unitToWad(9)
132      );
133
134      IERC20Upgradeable(asset).safeTransfer(protocolConfig.treasury, withdrawalFee);
135  }
136
137  if (localReserve.ext.bonusPool != address(0)) {
138      uint256 nextUserReserveBalance = IUserData(protocolConfig.userData).getUserReserve(user, asset, false);
139      IBonusPool(localReserve.ext.bonusPool).updatePoolUser(asset, user, nextUserReserveBalance);
140  }
141
142  reserve.postUpdateReserveData();
143
144  IERC20Upgradeable(asset).safeTransfer(user, amount - withdrawalFee);
145
146  emit WithdrawnReserve(user, asset, localReserve.ext.reinvestment, amount - withdrawalFee);
147  }
148

```

Figure 42 Source code of *executeWithdrawReserve* function(Fixed)

```

1074 function _repayShort(address user, address asset, uint256 amount) internal {
1075     _reserveConfig[asset].updateIndex();
1076
1077     (uint256 positionAbsoluteAmount,) = GeneralLogic.getUserPosition(
1078         assetConfig[asset],
1079         _userPosition[user][asset],
1080         _reserveConfig[asset].getBorrowIndex()
1081     );
1082
1083     // cap amount to max repayable
1084     if (amount > positionAbsoluteAmount) {
1085         amount = positionAbsoluteAmount;
1086     }
1087
1088     ValidationLogic.validateRepayShort(
1089         userLastTradeBlock[user],
1090         DataTypes.ValidateRepayShortParams(
1091             user,
1092             asset,
1093             amount
1094         )
1095     );
1096
1097     IERC20Upgradeable(asset).safeTransferFrom(user, address(this), amount);
1098
1099     TradeLogic.executeLonging(
1100         _reserveConfig[asset],
1101         assetConfig[asset],
1102         _userPosition[user][asset],
1103         userConfig[user],
1104         treasury,
1105         amount,
1106         true
1107     );
1108
1109     emit RepaidShort(user, asset, amount);

```

Figure 43 Source code of `_repayShort` function(Fixed)

[LeverFi-17] Improperly designed `_depositCollateral` function

Severity Level	Low
Type	Business Security
Lines	Ledger.sol#L914-916
Description	In the <code>_depositCollateral</code> function, when verifying whether the stake meets the requirements, the current user's share (<code>currUserDepositShareRay</code>) plus the amount is incorrectly used to determine whether the minimum stake amount is met.

```

908  function _depositCollateral(address user, address asset, address reinvestment, uint256 amount) internal {
909      uint256 collateralPoolId = collateralPoolList[asset][reinvestment];
910      DataTypes.AssetConfig memory assetConfigCache = assetConfig[asset];
911      DataTypes.CollateralConfig storage collateral = _collateralConfig[collateralPoolId];
912      uint256 unit = assetConfigCache.decimals;
913
914      uint256 currUserDepositShareRay = userCollateralShareRay[user][collateralPoolId];
915
916      ValidationLogic.validateDepositCollateral(collateral, assetConfigCache, userLastTradeBlock[user], amount, currUserDepositShareRay);
917  }

```

Figure 44 Source code of `_depositCollateral` function(Unfixed)

Recommendations	It is recommended to use the current user's stake amount.
Status	Fixed.

```

939  function _depositCollateral(address user, address asset, address reinvestment, uint256 amount) internal {
940      uint256 collateralPoolId = collateralPoolList[asset][reinvestment];
941      DataTypes.AssetConfig memory assetConfigCache = assetConfig[asset];
942      DataTypes.CollateralConfig storage collateral = _collateralConfig[collateralPoolId];
943      uint256 unit = assetConfigCache.decimals;
944
945      uint256 currUserDepositShareRay = userCollateralShareRay[user][collateralPoolId];
946
947      uint256 currCollateralBalance;
948      // is first deposit
949      if (currUserDepositShareRay == 0) {
950          userConfig[user].setUsingCollateral(collateralPoolId, true);
951      } else {
952          currCollateralBalance = ShareBaseAccounting.getAbsoluteAmount(
953              currUserDepositShareRay,
954              collateral.scaledTotalSupplyRay,
955              GeneralLogic.getCollateralSupply(collateral).unitToRay(unit)
956          ).rayToUnit(unit);
957      }
958
959      ValidationLogic.validateDepositCollateral(collateral, assetConfigCache, userLastTradeBlock[user], amount, currCollateralBalance);
960  }

```

Figure 45 Source code of `_depositCollateral` function(Fixed)

[LeverFi-18] The *validateTrade* function check error

Severity Level	Low
Type	Business Security
Lines	ValidationLogic.sol#L170-174
Description	In the <i>validateTrade</i> function, the state of reserve is repeatedly judged, and the state of assetConfig should be judged here.

```

162     function validateTrade(
163         mapping(address => DataTypes.ReserveConfig) storage reserve,
164         mapping(address => DataTypes.AssetConfig) storage assetConfig,
165         mapping(address => DataTypes.UserPosition) storage userPosition,
166         uint256 userLastTradeBlock,
167         DataTypes.ValidateTradeParams memory params
168     ) external view {
169         ValidateTradeVars memory vars;
170         require(reserve[params.shortAsset].state == DataTypes.AssetState.Active, "selling asset not active");
171         require(reserve[params.longAsset].state == DataTypes.AssetState.Active, "buying asset not active");
172         require(reserve[params.shortAsset].state == DataTypes.AssetState.Active, "selling asset not active");
173         require(reserve[params.longAsset].state == DataTypes.AssetState.Active, "buying asset not active");
174         require(userLastTradeBlock != block.number, "cannot execute trade with other actions");
175         require(params.tradeAmount != 0, "amount is zero");
176         if (userPosition[params.shortAsset]._type == DataTypes.PositionType.Short) {
177             vars.amountToShort = params.tradeAmount;

```

Figure 46 Source code of *validateTrade* function(Unfixed)

Recommendations It is recommended to modify reserve to assetConfig.

Status Fixed.

```

160     /**
161     */
162     function validateTrade(
163         mapping(address => DataTypes.ReserveConfig) storage reserve,
164         mapping(address => DataTypes.AssetConfig) storage assetConfig,
165         mapping(address => DataTypes.UserPosition) storage userPosition,
166         uint256 userLastTradeBlock,
167         DataTypes.ValidateTradeParams memory params
168     ) external view {
169         ValidateTradeVars memory vars;
170         require(assetConfig[params.shortAsset].state == DataTypes.AssetState.Active, "selling asset not active");
171         require(assetConfig[params.longAsset].state == DataTypes.AssetState.Active, "buying asset not active");
172         require(reserve[params.shortAsset].state == DataTypes.AssetState.Active, "selling asset not active");
173         require(reserve[params.longAsset].state == DataTypes.AssetState.Active, "buying asset not active");
174         require(userLastTradeBlock != block.number, "cannot execute trade with other actions");
175         require(params.tradeAmount != 0, "amount is zero");
176         if (userPosition[params.shortAsset]._type == DataTypes.PositionType.Short) {
177             vars.amountToShort = params.tradeAmount;
178         } else {
179             if (userPosition[params.shortAsset].amount < params.tradeAmount) {

```

Figure 47 Source code of *validateTrade* function(Fixed)

[LeverFi-19] The *configureCollateral* function cannot modify reinvestment

Severity Level	Low
Type	Business Security
Lines	Ledger.sol#
Description	Because reinvestment cannot be set in the <i>configureCollateral</i> function, but there is a function for reinvesting new reinvestment in the contract, the reinvestment function will not be available.

```

361     function initializeCollateral(
362         address asset,
363         address reinvestment,
364         uint256 feeMantissa,
365         uint256 ltv,
366         uint256 minDeposit
367     ) external onlyOperator {
368         require(collateralPoolList[asset][reinvestment] == 0, "already initialized");
369
370         collateralPoolLength++;
371         collateralPoolList[asset][reinvestment] = collateralPoolLength;
372
373         DataTypes.CollateralConfig storage collateral = _collateralConfig[collateralPoolLength];
374
375         configureCollateral(asset, reinvestment, feeMantissa, ltv, minDeposit, DataTypes.AssetState.Active);
376
377         collateral.poolId = collateralPoolLength;
378         collateral.asset = asset;
379         collateral.reinvestment = reinvestment;
380
381         emit InitializedCollateral(collateral.poolId, asset, reinvestment);
382     }
383

```

Figure 48 Source code of *initializeCollateral* function

```

393     function configureCollateral(
394         address asset,
395         address reinvestment,
396         uint256 feeMantissa,
397         uint256 ltv,
398         uint256 minDeposit,
399         DataTypes.AssetState state
400     ) public onlyOperator {
401         DataTypes.CollateralConfig storage collateral = _collateralConfig[collateralPoolList[asset][reinvestment]];
402
403         collateral.feeMantissa = feeMantissa;
404         collateral.ltv = ltv;
405         collateral.minBalance = minDeposit;
406         collateral.state = state;
407
408         emit ConfiguredCollateral(asset, reinvestment, feeMantissa, ltv, minDeposit, state);
409     }
410

```

Figure 49 Source code of *configureCollateral* function

```

825     function reinvestCollateralSupply(address asset, address reinvestment) external onlyOperator {
826
827
828         uint256 poolId = collateralPoolList[asset][reinvestment];
829         DataTypes.CollateralConfig storage collateralConfig = _collateralConfig[poolId];
830
831         require(collateralConfig.state == DataTypes.AssetState.Disabled, "state is not disabled");
832         require(collateralConfig.reinvestment != address(0), "reinvestment is not set");
833
834         IERC20Upgradeable(asset).safeApprove(collateralConfig.reinvestment, collateralConfig.liquidSupply);
835         IReinvestment(collateralConfig.reinvestment).invest(collateralConfig.liquidSupply);
836
837         emit ReinvestedCollateralSupply(asset, reinvestment, collateralConfig.liquidSupply);
838
839         collateralConfig.liquidSupply = 0;
840     }
841

```

Figure 50 Source code of *reinvestCollateralSupply* function

Recommendations It is recommended to add `collateralPoolList[asset][new_reinvestment] = collateralPoolList[asset][reinvestment]; del collateralPoolList[asset][reinvestment]` to the `configureCollateral` function, and then update the reinvestment in the new collateral. When setting, you should also judge whether `collateralPoolList[asset][new_reinvestment]` has been set. Of course, when setting, also judge the state, which can only be set when Disabled.

Status

Fixed.

```

274
275
276
277
278
279
280
281
282
function setCollateralReinvestment(uint256 pid, address newReinvestment) external onlyConfigurator {
    DataTypes.CollateralData storage collateral = collaterals[pid];
    require(collateral.asset != address(0), Errors.POOL_NOT_INITIALIZED);
    collateralList[collateral.asset][newReinvestment] = pid;
    delete collateralList[collateral.asset][collateral.reinvestment];
    collateral.reinvestment = newReinvestment;
}

```

Figure 51 Source code of *setCollateralReinvestment* function(Fixed)

[LeverFi-20] Improper design when settlement user awards

Severity Level	Low
Type	Business Security
Lines	ConvexReinvestmentLogic.sol#L168-170
Description	During the settlement of user rewards, it did not receive the current reward in time, resulting in small rewards for user settlement.

```

168 function _checkpoint(address user, uint256 currBalance) internal {
169     for (uint256 i = 0; i < rewardLength(); i++) {
170         (
171             RewardConfig memory reward,
172             UserReward memory userReward
173         ) = _calculateRewards(user, currBalance, rewards(i), rewardOfInternal(user, i), false);
174
175         setRewards(i, reward);
176         setRewardOf(user, i, userReward);
177     }
178 }

```

Figure 52 Source code of `_checkpoint` function(Unfixed)

Recommendations	It is recommended to receive the reward of the pool before settle the user reward.
Status	Fixed.

```

161 function _checkpoint(address user, uint256 currBalance) internal {
162     IConvexRewards(rewardPool()).getReward(address(this), true);
163
164     for (uint256 i = 0; i < rewardLength(); i++) {
165         (
166             RewardConfig memory reward,
167             UserReward memory userReward
168         ) = _calculateRewards(user, currBalance, rewards(i), rewardOfInternal(user, i), false);
169
170         setRewards(i, reward);
171         setRewardOf(user, i, userReward);
172     }
173 }

```

Figure 53 Source code of `_checkpoint` function(Fixed)

[LeverFi-21] lack of judgment on value

Severity Level	Info
Type	Business Security
Lines	Ledger.sol#L137-161
Description	The following function lacks judgment on the input value, resulting in any value can be set.

```

136  */
137  function setLeverageFactor(uint256 leverageFactor_) external onlyOperator {
138      emit LeverageFactorUpdated(leverageFactor, leverageFactor_);
139
140      leverageFactor = leverageFactor_;
141  }
142
143  /**
144   * @notice Setter for tradeFeeMantissa
145   * @param tradeFeeMantissa_ The new tradeFeeMantissa
146   */
147  function setTradeFee(uint256 tradeFeeMantissa_) external onlyOperator {
148      emit TradeFeeUpdated(tradeFeeMantissa, tradeFeeMantissa_);
149
150      tradeFeeMantissa = tradeFeeMantissa_;
151  }
152
153  /**
154   * @notice Setter for liquidationRatioMantissa
155   * @param liquidationRatioMantissa_ The new liquidationRatioMantissa
156   */
157  function setLiquidationRatio(uint256 liquidationRatioMantissa_) external onlyOperator {
158      emit LiquidationRatioUpdated(liquidationRatioMantissa, liquidationRatioMantissa_);
159
160      liquidationRatioMantissa = liquidationRatioMantissa_;
161  }
162

```

Figure 54 Source code of related functions(Unfixed)

```

118  function initialize(
119      address treasury_,
120      uint256 leverageFactor_,
121      uint256 liquidationRatioMantissa_,
122      uint256 tradeFeeMantissa_
123  ) public initializer {
124      treasury = treasury_;
125      leverageFactor = leverageFactor_;
126      liquidationRatioMantissa = liquidationRatioMantissa_;
127      tradeFeeMantissa = tradeFeeMantissa_;
128
129      _grantRole(OPERATOR_ROLE, msg.sender);
130      _grantRole(LIQUIDATE_EXECUTOR, msg.sender);
131  }
132

```

Figure 55 Source code of *initialize* function(Unfixed)

Recommendations	It is recommended to judge the range of the input value.
Status	Fixed.


```

137  */
138  function setLeverageFactor(uint256 leverageFactor_) external onlyOperator {
139      require(leverageFactor_ >= 1e18, "leverage factor cannot be lower than 1x");
140      require(leverageFactor_ <= 10e18, "leverage factor cannot be lower than 10x");
141
142      emit LeverageFactorUpdated(leverageFactor, leverageFactor_);
143
144      leverageFactor = leverageFactor_;
145  }
146
147  /**
148   * @notice Setter for tradeFeeMantissa
149   * @param tradeFeeMantissa_ The new tradeFeeMantissa
150   */
151  function setTradeFee(uint256 tradeFeeMantissa_) external onlyOperator {
152      require(tradeFeeMantissa_ <= 0.1e18, "fee cannot be more than 10%");
153
154      emit TradeFeeUpdated(tradeFeeMantissa, tradeFeeMantissa_);
155
156      tradeFeeMantissa = tradeFeeMantissa_;
157  }
158
159  /**
160   * @notice Setter for liquidationRatioMantissa
161   * @param liquidationRatioMantissa_ The new liquidationRatioMantissa
162   */
163  function setLiquidationRatio(uint256 liquidationRatioMantissa_) external onlyOperator {
164      require(liquidationRatioMantissa_ >= 0.5e18, "fee cannot be less than 50%");
165      require(liquidationRatioMantissa_ <= 0.9e18, "fee cannot be more than 90%");
166
167      emit LiquidationRatioUpdated(liquidationRatioMantissa, liquidationRatioMantissa_);
168
169      liquidationRatioMantissa = liquidationRatioMantissa_;
170  }
171

```

Figure 56 Source code of related functions(Fixed)

```

119  function initialize(
120      address treasury_,
121      uint256 leverageFactor_,
122      uint256 liquidationRatioMantissa_,
123      uint256 tradeFeeMantissa_
124  ) public initializer {
125      require(leverageFactor_ >= 1e18, "leverage factor cannot be lower than 1x");
126      require(leverageFactor_ <= 10e18, "leverage factor cannot be lower than 10x");
127      require(liquidationRatioMantissa_ >= 0.5e18, "fee cannot be less than 50%");
128      require(liquidationRatioMantissa_ <= 0.9e18, "fee cannot be more than 90%");
129      require(tradeFeeMantissa_ <= 0.1e18, "fee cannot be more than 10%");
130
131      treasury = treasury_;
132      leverageFactor = leverageFactor_;
133      liquidationRatioMantissa = liquidationRatioMantissa_;
134      tradeFeeMantissa = tradeFeeMantissa_;
135
136      _setRoleAdmin(OPERATOR_ROLE, OPERATOR_ROLE);
137      _setRoleAdmin(LIQUIDATE_EXECUTOR, OPERATOR_ROLE);
138
139      _grantRole(OPERATOR_ROLE, msg.sender);
140      _grantRole(LIQUIDATE_EXECUTOR, msg.sender);
141  }

```

Figure 57 Source code of *initialize* function(Fixed)

[LeverFi-22] The *getUserLiquidity* function design flaw

Severity Level	Info
Type	Business Security
Lines	GeneralLogic.sol#L185-188
Description	When calculating availableLeverageUsd, whether vars.pnlUsd>0 or vars.pnlUsd<0, should use $(\text{int256}(\text{vars.totalCollateralUsdPostLtv}) + \text{vars.pnlUsd}) * \text{int256}(\text{params.leverageFactor}) / \text{int256}(1\text{e}18) - \text{int}(\text{vars.totalShortUsd})$), because the part of the revenue can still be used as part of the user.

```

176     }
177     }
178     vars.userConfigCache.collateral = vars.userConfigCache.collateral >> 1;
179     vars.userConfigCache.position = vars.userConfigCache.position >> 1;
180     vars.i++;
181 }
182 vars.pnlUsd = int256(vars.totalLongUsd) - int256(vars.totalShortUsd);
183 vars.isLiquidatable = (int256(vars.totalCollateralUsdPreltv.wadMul(params.liquidationRatioMantissa)) + vars.pnlUsd) < 0;
184 vars.totalLeverageUsd = vars.totalCollateralUsdPostLtv.wadMul(params.leverageFactor);
185 vars.availableLeverageUsd = vars.pnlUsd > 0
186 ? (int256(vars.totalCollateralUsdPostLtv) * int256(params.leverageFactor) / int256(1e18) - int(vars.totalShortUsd)
187 : (int256(vars.totalCollateralUsdPostLtv) + vars.pnlUsd) * int256(params.leverageFactor) / int256(1e18) - int(vars.totalShortUsd);
188 return (
189     vars.totalCollateralUsdPreltv,
190     vars.totalCollateralUsdPostLtv,
191     vars.totalLongUsd,
192     vars.totalShortUsd,
193     vars.pnlUsd,
194     vars.totalLeverageUsd,
195     vars.availableLeverageUsd,
196     vars.isLiquidatable
197 );
198 }
199
200

```

Figure 58 Source code of *getUserLiquidity* function(Unfixed)

Recommendations	It is recommended to treat the revenue part as the part used by the user.
Status	Fixed.

```

180     }
181     }
182     vars.pnlUsd = int256(vars.totalLongUsd) - int256(vars.totalShortUsd);
183     vars.isLiquidatable = (int256(vars.totalCollateralUsdPreltv.wadMul(params.liquidationRatioMantissa)) + vars.pnlUsd) < 0;
184     vars.totalLeverageUsd = vars.totalCollateralUsdPostLtv.wadMul(params.leverageFactor);
185     vars.availableLeverageUsd = (int256(vars.totalCollateralUsdPostLtv) + vars.pnlUsd) * int256(params.leverageFactor) / int256(1e18) - int(vars.totalShortUsd);
186     return (
187         vars.totalCollateralUsdPreltv,
188         vars.totalCollateralUsdPostLtv,
189         vars.totalLongUsd,
190         vars.totalShortUsd,
191         vars.pnlUsd,
192         vars.totalLeverageUsd,
193         vars.availableLeverageUsd,
194         vars.isLiquidatable
195     );
196 }
197
198

```

Figure 59 Source code of *getUserLiquidity* function(Fixed)

[LeverFi-23] Lack of judgment on whether to add assets

Severity Level	Info
Type	Business Security
Lines	Ledger.sol#L175-197,244-275
Description	In the <i>initializeAsset</i> function of the Ledger contract, it is not determined whether the asset has been added. If the operator is not set properly, it will cause an error in the <i>assetList</i> record. Similarly, the <i>initializeCollateral</i> function should also determine whether the <i>collateralPoolList[asset][reinvestment]</i> has been added.

```

175 function initializeAsset(
176     address asset,
177     uint256 decimals,
178     address longReinvestment,
179     ISwapAdapter swapAdapter,
180     IPriceOracleGetter oracle,
181     DataTypes.AssetState state,
182     DataTypes.AssetMode mode
183 ) external onlyOperator {
184     assetLength++;
185     assetList[assetLength] = asset;
186
187     configureAsset(
188         asset,
189         decimals,
190         longReinvestment,
191         swapAdapter,
192         oracle,
193         state,
194         mode
195     );
196
197     emit InitializedAssetConfig(assetLength, asset);
198
199

```

Figure 60 Source code of *initializeAsset* function(Unfixed)

```

334 function initializeCollateral(
335     address asset,
336     address reinvestment,
337     uint256 feeMantissa,
338     uint256 ltv,
339     uint256 minDeposit
340 ) external onlyOperator {
341     collateralPoolLength++;
342     collateralPoolList[asset][reinvestment] = collateralPoolLength;
343
344     DataTypes.CollateralConfig storage collateral = _collateralConfig[collateralPoolLength];
345
346     configureCollateral(asset, reinvestment, feeMantissa, ltv, minDeposit, DataTypes.AssetState.Active);
347
348     collateral.poolId = collateralPoolLength;
349     collateral.asset = asset;
350     collateral.reinvestment = reinvestment;
351
352     emit InitializedCollateral(collateral.poolId, asset, reinvestment);
353

```

Figure 61 Source code of *initializeCollateral* function(Unfixed)

Recommendations It is recommended to determine whether the asset has been added.

Status Fixed.

```

183 ~/
184 function initializeAsset(
185     address asset,
186     uint256 decimals,
187     address longReinvestment,
188     ISwapAdapter swapAdapter,
189     IPriceOracleGetter oracle,
190     DataTypes.AssetState state,
191     DataTypes.AssetMode mode
192 ) external onlyOperator {
193     require(assetConfig[asset].assetId == 0, "already initialized");
194
195     assetLength++;
196     assetList[assetLength] = asset;
197
198     configureAsset(
199         asset,
200         decimals,
201         longReinvestment,
202         swapAdapter,
203         oracle,
204         state,
205         mode
206     );
207
208     emit InitializedAssetConfig(assetLength, asset);
209 }
210

```

Figure 62 Source code of *initializeAsset* function(Fixed)

```

361 ~/
362 function initializeCollateral(
363     address asset,
364     address reinvestment,
365     uint256 feeMantissa,
366     uint256 ltv,
367     uint256 minDeposit
368 ) external onlyOperator {
369     require(collateralPoolList[asset][reinvestment] == 0, "already initialized");
370
371     collateralPoolLength++;
372     collateralPoolList[asset][reinvestment] = collateralPoolLength;
373
374     DataTypes.CollateralConfig storage collateral = _collateralConfig[collateralPoolLength];
375
376     configureCollateral(asset, reinvestment, feeMantissa, ltv, minDeposit, DataTypes.AssetState.Active);
377
378     collateral.poolId = collateralPoolLength;
379     collateral.asset = asset;
380     collateral.reinvestment = reinvestment;
381
382     emit InitializedCollateral(collateral.poolId, asset, reinvestment);
383 }
384

```

Figure 63 Source code of *initializeCollateral* function(Fixed)

[LeverFi-24] Admin permission not initialized

Severity Level	Info
Type	Business Security
Lines	Ledger.sol#L118-130
Description	In the initialize function, there is no initial admin permission, which will result in the subsequent failure to change the permissions of OPERATOR_ROLE and LIQUIDATE_EXECUTOR.

```

117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
27
```

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 itself, 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

Contact@beosin.com

