Byte Masons - iUSD
Audit Report

Version 1.1

Zigtur

July 25, 2024

Byte Masons - iUSD - Audit Report

Zigtur

July 25, 2024

Prepared by: Zigtur

Table of Contents

Introduction

Disclaimer

A smart contract security review cannot guarantee the complete absence of vulnerabilities. This effort, bound by time, resources, and expertise, aims to identify as many security issues as possible. However, there is no assurance of 100% security post-review, nor is there a guarantee that the review will uncover all potential problems in the smart contracts. It is highly recommended to conduct subsequent security reviews, implement bug bounty programs, and perform on-chain monitoring.

About Zigtur

Zigtur is an independent blockchain security researcher dedicated to enhancing the security of the blockchain ecosystem. With a history of identifying numerous security vulnerabilities across various protocols in public audit contests and private audits, **Zigtur** strives to contribute to the safety and reliability of blockchain projects through meticulous security research and reviews. Explore previous work here or reach out on X @zigtur.

About iUSD stablecoin

iUSD stablecoin is a CDP stablecoin protocol.

It is a fork from Ethos Reserve.

Security Assessment Summary

Review commit hash - 22fad1a

Fixes review commit hash - 4f5d8d8

Deployment chains

- All EVM chains/rollups

Dependencies

The codebase at commit 22fad1a relies on Byte-Masons/vault-v2 at commit 2f30c6c.

Scope

This audit focuses on modifications made to an Ethos V2 fork. The newly implemented features are:

- Reaper vault (ERC4626) are now used as collateral
- Rehypothecation removal
- `LQTYStaking` removal
- Rewards system
- Support of multiple DEX for `Leverager`

The following smart contracts are in scope of the review:

- ActivePool.sol
- BorrowerOperations.sol
- BorrowerHelper.sol
- CollateralConfig.sol
- CollSurplusPool.sol
- DefaultPool.sol
- GasPool.sol
- HintHelpers.sol
- Leverager.sol
- LiquidationHelper.sol
- LUSDToken.sol
- Migrations.sol
- MultiTroveGetter.sol
- PriceFeed.sol
- RedemptionHelper.sol
- RewarderManager.sol
- SortedTroves.sol
- StabilityPool.sol
- TroveManager.sol
- LQTY/CommunityIssuance.sol
- Dependencies/UsingTellor.sol
- Dependencies/LiquityMath.sol
- Dependencies/LiquityBase.sol
- Dependencies/Ownable.sol
- Dependencies/TellorCaller.sol
- Dependencies/Context.sol
- Dependencies/CheckContract.sol
- Dependencies/BaseMath.sol

- Proxy/BorrowerOperationsScript.sol
- Proxy/BorrowerWrappersScript.sol
- Proxy/TokenScript.sol
- Proxy/TroveManagerScript.sol
- Proxy/StabilityPoolScript.sol
- Proxy/ERC20TransferScript.sol

Risk Classification

	Impact: High	**Impact:** Medium	**Impact:** Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Issues

HIGH-01 - Troves will never be adjustable through `BorrowerHelper`

Description

Scope:

- BorrowerOperations.sol#L507-L508

The `BorrowerHelper` allows users to adjust their troves by calling `BorrowerOperations.adjustTroveFor`.

However, the `_adjustTrove` internal function will check that the caller is either the borrower or the `Leverager` contract. It will always revert when `BorrowerHelper` is the caller.

Impact

Troves are not adjustable through `BorrowerHelper`.

Code snippet

`BorrowerOperations._adjustTrove` ensures that `msg.sender` is the borrower or the `Leverager` contract.

```
507     // Confirm the operation is a borrower adjusting their own trove (possibly
    ↪   through the Leverager)
508     assert(msg.sender == params._borrower || msg.sender == leveragerAddress);
```

This will always revert when it is called from the `BorrowerHelper`.

Proof of Concept

A patch is given in Appendix to import the PoC.

Recommendation

The `assert` statement should succeed when the `borrowerHelper` address is the caller. The following code could be used.

```
    // Confirm the operation is a borrower adjusting their own trove (possibly
    ↪  through the Leverager)
    assert(
      msg.sender == params._borrower ||
      msg.sender == leveragerAddress ||
      msg.sender == helperAddress
    );
```

A patch is given in Appendix to apply this recommendation. It also replaces `assert` with `require` to revert with a string.

Resolution

Byte Masons team: Fixed. Provided patch applied.

Zigtur: Fix reviewed and approved.

HIGH-02 - Users can lose withdrawn collaterals when withdrawing collateral and increasing debt in a single `adjustTrove` call

Description

Scope:

- BorrowerHelper.sol#L113-L117

The `BorrowerHelper.adjustTrove` function allows the user to interact with his trove. He can increase debt, decrease debt or withdraw collateral through this function.

However, the `BorrowerHelper` contract doesn't transfer the collateral to the user when this user increases debt and withdraws collateral in the same call.

Note that this behavior is allowed in the `BorrowerOperations` called by `BorrowerHelper`.

Impact

User will lose withdrawn collateral.

Code snippet

The `BorrowerHelper.adjustTrove` function handles debt increase or collateral withdrawal, but not both in the same call.

```
113     if (_isDebtIncrease) {
114         lusdToken.safeTransfer(msg.sender, lusdToken.balanceOf(address(this))); // @POC: transfer LUSD to user if debt has increased (LUSD borrowing)
115     } else {
116         _withdrawAndTransfer(_collateral);  // @POC: If debt increases, else is never executed and collateral is not sent to user
117     }
```

Proof of Concept

A Foundry unit test file is given in Appendix.

Note: For this PoC to work, two other issues needed to be fixed (`HIGH-01` and `MEDIUM-02`). The patch given to import the PoC modifies the codebase to fix these two issues.

Recommendation

Withdrawn collateral should be sent to the user even if `_isDebtIncrease == true`.

```
    if (_isDebtIncrease) {
        lusdToken.safeTransfer(msg.sender, lusdToken.balanceOf(address(this)));
    }
    if (_collWithdrawal != 0) {
        _withdrawAndTransfer(_collateral);
    }
```

A patch is given in Appendix to apply this recommendation.

Resolution

Byte Masons team: Fixed. Provided patch applied.

Zigtur: Fix reviewed and approved.

HIGH-03 - Fetching price will return asset price instead of share price when oracles work as expected

Description

Scope:

- PriceFeed.sol#L605-L607
- PriceFeed.sol#L612-L614
- PriceFeed.sol#L585-L601

The `_storeChainlinkPrice` and `_storeTellorPrice` functions are responsible for scaling returning the correct collateral price from an oracle response. The support of vaults as collaterals required modifications of the `PriceFeed` to convert an asset price to a share price. This price conversion has been implemented in `_storePrice` which is called by both `_storeChainlinkPrice` and `_storeTellorPrice`.

However, both functions ignore the share price returned by `_storePrice`. They both returned the underlying asset price instead of the share price.

Code snippet

Note: The following shows `_storeChainlinkPrice`, but `_storeTellorPrice` is also affected.

`_storeChainlinkPrice` calls `_storePrice` but returns `scaledChainlinkPrice` instead of the `_storePrice` returned value.

```
610  function _storeChainlinkPrice(address _collateral, ChainlinkResponse memory
     ↪  _chainlinkResponse) internal returns (uint) {
611      uint scaledChainlinkPrice =
         ↪  _scaleChainlinkPriceByDigits(uint256(_chainlinkResponse.answer),
         ↪  _chainlinkResponse.decimals);
612      _storePrice(_collateral, scaledChainlinkPrice); // @POC: Share price is
         ↪  ignored
613
614      return scaledChainlinkPrice; // @POC: Asset price is returned instead
615  }
```

Recommendation

`_storeChainlinkPrice` and `_storeTellorPrice` should return the value returned by `_storePrice`.

```
function _storeChainlinkPrice(address _collateral, ChainlinkResponse memory
↪  _chainlinkResponse) internal returns (uint) {
    uint scaledChainlinkPrice =
    ↪  _scaleChainlinkPriceByDigits(uint256(_chainlinkResponse.answer),
    ↪  _chainlinkResponse.decimals);
    return _storePrice(_collateral, scaledChainlinkPrice);
}
```

A patch is given in Appendix to apply this recommendation.

Resolution

Byte Masons team: Fixed. Provided patch applied.

Zigtur: Fix reviewed and approved.

HIGH-04 - Rewarder implementation must not make external calls

Description

Scope:

- RewarderManager.sol#L57-L100

The `Rewarder` contract implementation is not known in the current state of the codebase.

However, this implementation **must never call external contract**, especially if an external user could manipulate this call target. This could lead to reentrancy issues.

Code snippet

For example, the `TroveManager.updateDebtAndCollAndStakesPostRedemption` function is given.

```solidity
    function updateDebtAndCollAndStakesPostRedemption(
        address _borrower,
        address _collateral,
        uint256 _newDebt,
        uint256 _newColl
    ) external override {
        _requireCallerIsRedemptionHelper();
        uint256 oldDebt = Troves[_borrower][_collateral].debt;
        uint256 oldColl = Troves[_borrower][_collateral].coll;
        rewarderManager.onDebtDecrease(_borrower, _collateral, oldDebt -
        ↪   _newDebt); // @POC: RewarderManager call
        rewarderManager.onCollDecrease(_borrower, _collateral, oldColl -
        ↪   _newColl); // @POC: RewarderManager call
        Troves[_borrower][_collateral].debt = _newDebt;
        Troves[_borrower][_collateral].coll = _newColl;
        _updateStakeAndTotalStakes(_borrower, _collateral);

```

As we can see, the two calls to `RewarderManager` are done before writing the `_newDebt` and `_newColl` values.

This could lead to a reentrancy exploit if the `Rewarder` implementation (called by `RewardManager`) allows reentrancy.

Recommendation

Design the `Rewarder` implementation such that it doesn't call untrusted addresses.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged.

MEDIUM-01 - `BorrowerHelper` can't be unpaused

Description

- BorrowerHelper.sol#L33-L35

The `BorrowerHelper` contract allows the owner to pause the contract through `pause()`.

However, once paused, the owner will never be able to unpause the contract as there is no `unpause()` function.

Recommendation

Implement an `unpause()` function.

A patch is given in Appendix to apply this recommendation.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged. Once paused, the `BorrowerHelper` will never be reusable. It will require deploying a new instance of the contract as `BorrowerHelper` is not upgradable. This will also require modification of the `BorrowerOperations` configuration to update the `helperAddress` value.

MEDIUM-02 - User can't increase their debt through `BorrowerHelper` without depositing assets

Description

Scope:

- BorrowerHelper.sol#L97-L99
- BorrowerHelper.sol#L125-L137
- Dependency: ReaperVaultV2.sol#L314

The `BorrowerHelper.adjustTrove` function allows the user to interact with his trove. He can increase debt, decrease debt or withdraw collateral through this function.

However, when a user wants to increase debt without depositing more collateral, a zero deposit will be made to `ReaperVaultV2`. This contract will revert because it doesn't handle zero deposit.

Impact

User can't increase debt through `BorrowerHelper` without depositing collateral.

Code snippet

`adjustTrove` will deposit asset to `ReaperVaultV2` if `_isDebtIncrease == true`. This is done by calling `_transferAndDeposit` which calls the vault even if amount is zero.

```
    function adjustTrove(
        ...
        uint _collTopUp,
        ...
    ) external whenNotPaused {
        if (_isDebtIncrease) {
            _collTopUp = _transferAndDeposit(_collateral, _collTopUp); // @POC:
             ↪  deposit `_collTopUp` amount
        }
        ...
    }

    function _transferAndDeposit(
        address _collateral,
        uint _collAmount
    ) private returns (uint shares) {
        IReaperVault vault = IReaperVault(_collateral);
        IERC20 asset = IERC20(vault.asset());

        asset.safeTransferFrom(msg.sender, address(this), _collAmount);
        asset.safeIncreaseAllowance(_collateral, _collAmount);
        shares = vault.deposit(_collAmount, address(this)); // @POC: Will deposit
        ↪  zero amount

        IERC20(vault).safeIncreaseAllowance(address(borrowerOperations), shares);
    }
```

Then, `ReaperVaultV2` will revert as it doesn't handle zero deposit.

```
    function _deposit(uint256 _amount, address _receiver) internal nonReentrant
    ↪  returns (uint256 shares) {
        require(!emergencyShutdown, "Cannot deposit during emergency shutdown");
        require(_amount != 0, "Invalid amount"); // @POC: revert on zero deposit
        require(balance() + _amount <= tvlCap, "Vault is full");
```

Proof of Concept

A Foundry unit test file is given in Appendix.

Note: For this PoC to work, another issue needed to be fixed (`HIGH-01`). The patch given to import the PoC modifies the codebase to fix this issue.

Recommendation

`_transferAndDeposit` should not call `ReaperVaultV2` to deposit zero amount. This can be done by adding a check in `adjustTrove` or in `_transferAndDeposit`.

```
    function adjustTrove(
        ...
        uint _collTopUp,
        ...
    ) external whenNotPaused {
        if (_collTopUp != 0) {
            _collTopUp = _transferAndDeposit(_collateral, _collTopUp); // @POC:
            ↪  deposit `_collTopUp` amount
        }
        ...
    }
```

A patch is given in Appendix to apply this recommendation.

Resolution

Byte Masons team: Fixed. Provided patch applied.

Zigtur: Fix reviewed and approved.

MEDIUM-03 - User can’t increase their collateral through `BorrowerHelper` without increasing their debt

Scope:

- BorrowerHelper.sol#L97-L99
- BorrowerOperations.sol#L435-L441
- BorrowerOperations.sol#L835-L848

Through `BorrowerHelper.adjustTrove`, a user may want to provide more collateral by setting `_collTopUp = amount` and `_isDebtIncrease`.

However, `BorrowerOperations` will revert with this configuration because `BorrowerHelper` did not take collateral from the user.

Code snippet

`BorrowerHelper.adjustTrove` transfers and deposit collateral only if the debt increase, which is not the case when only providing collateral.

```
    function adjustTrove(
        ...
        uint _collTopUp,
        ...
    ) external whenNotPaused {
        if (_isDebtIncrease) {
            _collTopUp = _transferAndDeposit(_collateral, _collTopUp); // @POC:
            ↪  deposit `_collTopUp` amount
        }
        ...
    }
```

Proof of Concept

A Foundry unit test file is given in Appendix.

Recommendation

`adjustTrove` should retrieve funds from the user when `_collTopUp` is not zero.

```
    function adjustTrove(
        ...
        uint _collTopUp,
        ...
    ) external whenNotPaused {
        if (_collTopUp != 0) {
            _collTopUp = _transferAndDeposit(_collateral, _collTopUp); // @POC:
             ↪  deposit `_collTopUp` amount
        }
        ...
    }
```

The same patch given for `MEDIUM-02` will apply this recommendation.

Resolution

Byte Masons team: Fixed. Provided patch applied.

Zigtur: Fix reviewed and approved.

MEDIUM-04 - Debt repayment with LUSD is not supported with `BorrowerHelper`

Description

Scope:

- BorrowerHelper.sol#L87-L118
- BorrowerOperations.sol#L557-L562
- BorrowerOperations.sol#L1004-L1013

The `BorrowerHelper.adjustTrove` function does not handle LUSD repayment. A LUSD repayment is triggered when `_LUSDChange > 0` and `_isDebtIncrease == false`.

However, `BorrowerHelper` does not transfer the LUSD tokens from the user to itself, so `BorrowerOperations` will revert because the helper doesn't have enough LUSD to make repayment.

Proof of Concept

A Foundry unit test file is given in Appendix.

Note: For this PoC to work, another issue needed to be fixed (`HIGH-01`). The patch given to import the PoC modifies the codebase to fix this issue.

Recommendation

When `_LUSDChange > 0` and `_isDebtIncrease == false`, `BorrowerHelper` should transfer LUSD from the caller and approve `BorrowerOperations` to make a LUSD repayment on user's behalf.

Resolution

Byte Masons team: Fixed. Patched following recommendation.

Zigtur: Fix reviewed and approved.

MEDIUM-05 - A failing rewarder contract will DOS the whole protocol

Description

Scope:

- RewarderManager.sol#L58-L100

Reward hooks were implemented in the `TroveManager` contract. Multiple calls can be made to the `RewarderManager` contract for handling rewards calculations. `RewarderManager` then calls all the configured rewarders.

The `Rewarder` implementation is not known in the current state of the codebase.

However, if the `Rewarder` is prone to revert for any reason (underflow/overflow, incorrect checks, …), the whole protocol would break.

Recommendation

`try/catch` could be implemented to avoid breaking the whole protocol **if and only if** rewards are not a critical feature in the protocol.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged.

LOW-01 - `Leverager` will not work with every versions of `ReaperSwapper`

Description

- Leverager.sol#L533
- Leverager.sol#L543
- Leverager.sol#L553
- Leverager.sol#L563
- main branch: ReaperSwapper.sol#L143-L163
- mode branch: ReaperSwapper.sol#L180

`Leverager` calls the `ReaperSwapper` swap functions with a boolean as the final argument. This boolean indicates if the call should revert or not on failing swaps.

However, the current codebase retrieves Byte-Masons/vault-v2 repository on main branch (commit 2f30c6c).. On this branch, the swap functions do not take a boolean.

The branch "mode" (commit 6cbaea2) implements this boolean.

Recommendation

The `Leverager` contract must be deployed with compatible `ReaperSwapper`.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged.

GAS-01 - Multiple operations can be unchecked in `LUSDToken`

Description

Scope:

- LUSDToken.sol

The `LUSDToken` has been updated for Solidity 8.

`unchecked` operations could be used in `_transfer` and `_burn` functions to consume less gas.

Recommendation

Use `unchecked` in `_transfer` and `_burn` when a `require` statement already guarantees that the operation will not underflow.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged.

INFO-01 - PUSH0 instruction is not supported by all chains

Description

The Solidity compiler version used for must be greater than or equal to `0.8.23`. This version includes the `PUSH0` instruction.

However, some chains such as Linea doesn't support this instruction yet.

Recommendation

Consider compiling with Solidity `0.8.19` to remove the `PUSH0` instruction for chains without `PUSH0` support.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged.

INFO-02 - `PriceFeed` contract is still using SafeMath library

Description

Scope:

- PriceFeed.sol#L25

`PriceFeed` contract is still using `SafeMath` with Solidity version 8.

Recommendation

Consider removing `SafeMath` and updating `PriceFeed` code.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged.

INFO-03 - Tests are missing/failing

Description

Multiple tests in the codebase are failing. For example, `testCloseTrove` in `BorrowerHelper.t.sol` fails.

Moreover, several functionalities are not tested. For example, `BorrowerHelper.adjustTrove` test is commented-out.

Recommendation

Consider reviewing the test suite to ensure sufficient quality.

Resolution

Byte Masons team: Proof of Concept tests from the audit have been imported.

Zigtur: Partially fixed.

INFO-04 - Unused internal function in `ActivePool`

Description

Scope:

- ActivePool.sol#L190-L195

The `_requireCallerIsOwnerOrCollateralConfig` internal function is never used in the `ActivePool` contract.

Recommendation

Consider removing `_requireCallerIsOwnerOrCollateralConfig`.

Resolution

Byte Masons team: Acknowledged.

Zigtur: Acknowledged.

Appendix

HIGH-01 - Fix patch

The following patch can be applied through `git apply` to import the recommended fix.

```
diff --git a/src/contracts/BorrowerOperations.sol
↪  b/src/contracts/BorrowerOperations.sol
index e86a38c..f06fa1a 100644
--- a/src/contracts/BorrowerOperations.sol
+++ b/src/contracts/BorrowerOperations.sol
@@ -505,7 +505,10 @@ contract BorrowerOperations is LiquityBase, Ownable,
↪  CheckContract, IBorrowerOpe
         _requireTroveisActive(contractsCache.troveManager, params._borrower,
         ↪  params._collateral);

         // Confirm the operation is a borrower adjusting their own trove
         ↪  (possibly through the Leverager)
-        assert(msg.sender == params._borrower || msg.sender == leveragerAddress);
+        require(
+            msg.sender == params._borrower || msg.sender == leveragerAddress ||
↪  msg.sender == helperAddress,
+            "BorrowerOperations: Untrusted caller"
+        );

         contractsCache.troveManager.applyPendingRewards(params._borrower,
         ↪  params._collateral);
```

HIGH-01 - Proof of Concept

The following patch imports a Foundry unit test to show the issue.

Note: the code leading to the issue was modified to revert with a string.

```
diff --git a/src/contracts/BorrowerOperations.sol
 ↪  b/src/contracts/BorrowerOperations.sol
index e86a38c..cafcceb 100644
--- a/src/contracts/BorrowerOperations.sol
+++ b/src/contracts/BorrowerOperations.sol
@@ -505,7 +505,8 @@ contract BorrowerOperations is LiquityBase, Ownable,
 ↪  CheckContract, IBorrowerOpe
         _requireTroveisActive(contractsCache.troveManager, params._borrower,
          ↪  params._collateral);

         // Confirm the operation is a borrower adjusting their own trove
          ↪  (possibly through the Leverager)
-        assert(msg.sender == params._borrower || msg.sender == leveragerAddress);
+        //assert(msg.sender == params._borrower || msg.sender ==
 ↪  leveragerAddress);
+        require(msg.sender == params._borrower || msg.sender == leveragerAddress,
 ↪  "POC: wrong message sender");

         contractsCache.troveManager.applyPendingRewards(params._borrower,
          ↪  params._collateral);

diff --git a/src/test/forge/BorrowerHelper.t.sol
 ↪  b/src/test/forge/BorrowerHelper.t.sol
index f307d11..f65151f 100644
--- a/src/test/forge/BorrowerHelper.t.sol
+++ b/src/test/forge/BorrowerHelper.t.sol
@@ -148,6 +148,36 @@ contract BorrowerHelperTest is Test {
         borrowerHelper.adjustTrove(address(iclVault), 0.005 ether, collTopUp,
          ↪  collWithdrawal, lusdChange, isDebtIncrease, address(0), address(0));
     }*/

+    function testPOCHigh01AdjustTrove(
+        uint targetCR,
+        uint newTargetCR
+    ) public {
+        uint MCR = collateralConfig.getCollateralMCR(address(iclVault));
+        targetCR = bound(targetCR, MCR, 5 ether);
+        newTargetCR = bound(newTargetCR, MCR, 5 ether);
+        uint256 collAmount = 1000 ether;
+
+        uint collTopUp = 10e18;
+        uint collWithdrawal = 0;
+        uint lusdChange = 10e18;
+        bool isDebtIncrease = true;
+
+
+        deal(address(icl), address(this), collAmount);
+        icl.approve(address(borrowerHelper), collAmount);
+
+        uint debt = collAmount * 1e18 / targetCR -
↪   borrowerOperations.LUSD_GAS_COMPENSATION();
+        debt -= troveManager.getBorrowingFeeWithDecay(debt);
+        borrowerHelper.openTrove(address(iclVault), collAmount, 0.005 ether,
↪   debt, address(0), address(0));
+
+
+        deal(address(icl), address(this), collTopUp);
+        icl.approve(address(borrowerHelper), collTopUp);
+
+        vm.expectRevert("POC: wrong message sender"); // @POC:
↪   `BorrowerOperations` will revert
+        borrowerHelper.adjustTrove(address(iclVault), 0.005 ether, collTopUp,
↪   collWithdrawal, lusdChange, isDebtIncrease, address(0), address(0));
+     }
+
     /*function testClaimCollateral() public {
         vm.prank(whale);
         borrowerOperations.withdrawColl(address(iclVault), 1e26 - 130 ether,
         ↪  address(0), address(0));
```

HIGH-02 - Fix patch

The following patch can be applied through `git apply` to import the recommended fix.

```
diff --git a/src/contracts/BorrowerHelper.sol b/src/contracts/BorrowerHelper.sol
index bbc0e37..fc4075e 100644
--- a/src/contracts/BorrowerHelper.sol
+++ b/src/contracts/BorrowerHelper.sol
@@ -112,7 +112,8 @@ contract BorrowerHelper is LiquityBase, Ownable, CheckContract
↪  {
         );
         if (_isDebtIncrease) {
             lusdToken.safeTransfer(msg.sender,
              ↪  lusdToken.balanceOf(address(this)));
-        } else {
+        }
+        if (_collWithdrawal != 0) {
             _withdrawAndTransfer(_collateral);
         }
     }
```

HIGH-02 - Proof of Concept

The following patch imports a Foundry unit test to show the issue. Execute `forge test --mt test_POC -vvv`.

Note: For this PoC to work, two other issues needed to be fixed (`HIGH-01` and `MEDIUM-02`). The patch given to import the PoC modifies the codebase to fix these two issues.

```
diff --git a/src/contracts/BorrowerHelper.sol b/src/contracts/BorrowerHelper.sol
index bbc0e37..224ebb2 100644
--- a/src/contracts/BorrowerHelper.sol
+++ b/src/contracts/BorrowerHelper.sol
@@ -94,7 +94,7 @@ contract BorrowerHelper is LiquityBase, Ownable, CheckContract {
         address _upperHint,
         address _lowerHint
     ) external whenNotPaused {
-        if (_isDebtIncrease) {
+        if (_collTopUp > 0) { // @POC: Fix MEDIUM-02
             _collTopUp = _transferAndDeposit(_collateral, _collTopUp);
         }
         borrowerOperations.adjustTroveFor(
diff --git a/src/contracts/BorrowerOperations.sol
 ↪ b/src/contracts/BorrowerOperations.sol
index e86a38c..01b5bac 100644
--- a/src/contracts/BorrowerOperations.sol
+++ b/src/contracts/BorrowerOperations.sol
@@ -505,7 +505,7 @@ contract BorrowerOperations is LiquityBase, Ownable,
 ↪ CheckContract, IBorrowerOpe
         _requireTroveisActive(contractsCache.troveManager, params._borrower,
          ↪ params._collateral);

         // Confirm the operation is a borrower adjusting their own trove
          ↪ (possibly through the Leverager)
-        assert(msg.sender == params._borrower || msg.sender == leveragerAddress);
+        assert(msg.sender == params._borrower || msg.sender == leveragerAddress
 ↪ || msg.sender == helperAddress); // @POC: Fix HIGH-01

         contractsCache.troveManager.applyPendingRewards(params._borrower,
          ↪ params._collateral);

diff --git a/src/test/forge/BorrowerHelper.t.sol
 ↪ b/src/test/forge/BorrowerHelper.t.sol
index f307d11..ecb8037 100644
--- a/src/test/forge/BorrowerHelper.t.sol
+++ b/src/test/forge/BorrowerHelper.t.sol
@@ -148,6 +148,41 @@ contract BorrowerHelperTest is Test {
         borrowerHelper.adjustTrove(address(iclVault), 0.005 ether, collTopUp,
          ↪ collWithdrawal, lusdChange, isDebtIncrease, address(0), address(0));
```

```
    }*/

+    function testPOCHigh02AdjustTrove(
+        uint newTargetCR,
+        bool isDebtIncrease
+    ) public {
+        uint MCR = collateralConfig.getCollateralMCR(address(iclVault));
+        uint targetCR = 5 ether;
+        newTargetCR = bound(newTargetCR, MCR, 5 ether);
+        uint256 collAmount = 10000 ether;
+
+        uint collTopUp = 0;
+        uint collWithdrawal = 20e18;
+        uint lusdChange = 10e18;
+        bool isDebtIncrease = true;
+
+
+        deal(address(icl), address(this), collAmount);
+        icl.approve(address(borrowerHelper), collAmount);
+
+        uint debt = collAmount * 1e17 / targetCR -
↪   borrowerOperations.LUSD_GAS_COMPENSATION();
+        debt -= troveManager.getBorrowingFeeWithDecay(debt);
+        borrowerHelper.openTrove(address(iclVault), collAmount, 0.005 ether,
↪   debt, address(0), address(0));
+
+
+        deal(address(icl), address(this), collTopUp);
+        icl.approve(address(borrowerHelper), collTopUp);
+
+        uint256 helperSharesBalance =
↪   iclVault.balanceOf(address(borrowerHelper));
+
+        borrowerHelper.adjustTrove(address(iclVault), 0.005 ether, collTopUp,
↪   collWithdrawal, lusdChange, isDebtIncrease, address(0), address(0));
+
+        uint256 helperSharesAfter = iclVault.balanceOf(address(borrowerHelper));
↪   // @POC: Balance of vault shares has increased due to collateral withdrawal
↪   not being sent back to user
+        uint256 helperSharesDiff  = helperSharesAfter - helperSharesBalance;
+        assert(helperSharesDiff == collWithdrawal); // @POC: Lost funds are 20e18
+    }
+
     /*function testClaimCollateral() public {
         vm.prank(whale);
         borrowerOperations.withdrawColl(address(iclVault), 1e26 - 130 ether,
             ↪  address(0), address(0));
```

HIGH-03 - Fix patch

The following patch can be applied through `git apply` to import the recommended fix.

```diff
diff --git a/src/contracts/PriceFeed.sol b/src/contracts/PriceFeed.sol
index b3b5a02..2ac85fb 100644
--- a/src/contracts/PriceFeed.sol
+++ b/src/contracts/PriceFeed.sol
@@ -602,16 +602,12 @@ contract PriceFeed is Ownable, CheckContract, BaseMath,
↪  IPriceFeed {

      function _storeTellorPrice(address _collateral, TellorResponse memory
      ↪  _tellorResponse) internal returns (uint) {
         uint scaledTellorPrice =
         ↪  _scaleTellorPriceByDigits(_tellorResponse.value);
-        _storePrice(_collateral, scaledTellorPrice);
-
-        return scaledTellorPrice;
+        return _storePrice(_collateral, scaledTellorPrice);
     }

     function _storeChainlinkPrice(address _collateral, ChainlinkResponse memory
     ↪  _chainlinkResponse) internal returns (uint) {
         uint scaledChainlinkPrice =
         ↪  _scaleChainlinkPriceByDigits(uint256(_chainlinkResponse.answer),
         ↪  _chainlinkResponse.decimals);
-        _storePrice(_collateral, scaledChainlinkPrice);
-
-        return scaledChainlinkPrice;
+        return _storePrice(_collateral, scaledChainlinkPrice);
     }

     function _storeAssetsPerShare(address _collateral, uint _assetsPerShare)
     ↪  internal {
```

MEDIUM-01 - Fix patch

The following patch can be applied through `git apply` to import the recommended fix.

```
diff --git a/src/contracts/BorrowerHelper.sol b/src/contracts/BorrowerHelper.sol
index bbc0e37..333f374 100644
--- a/src/contracts/BorrowerHelper.sol
+++ b/src/contracts/BorrowerHelper.sol
@@ -34,6 +34,10 @@ contract BorrowerHelper is LiquityBase, Ownable, CheckContract
↪  {
         paused = true;
     }

+    function unpause() external onlyOwner {
+        paused = false;
+    }
+
     function setAddresses(
         address _borrowerOperationsAddress,
         address _troveManagerAddress,
```

MEDIUM-02 - Fix patch

The following patch can be applied through `git apply` to import the recommended fix.

```diff
diff --git a/src/contracts/BorrowerHelper.sol b/src/contracts/BorrowerHelper.sol
index bbc0e37..a6513b7 100644
--- a/src/contracts/BorrowerHelper.sol
+++ b/src/contracts/BorrowerHelper.sol
@@ -94,7 +94,7 @@ contract BorrowerHelper is LiquityBase, Ownable, CheckContract {
         address _upperHint,
         address _lowerHint
     ) external whenNotPaused {
-        if (_isDebtIncrease) {
+        if (_collTopUp != 0) {
             _collTopUp = _transferAndDeposit(_collateral, _collTopUp);
         }
         borrowerOperations.adjustTroveFor(
```

MEDIUM-02 - Proof of Concept

The following patch imports a Foundry unit test to show the issue. Execute `forge test --mt test_POC -vvv`.

Note: For this PoC to work, another issue needed to be fixed (`HIGH-01`). The patch given to import the PoC modifies the codebase to fix this issue.

```
diff --git a/src/test/forge/BorrowerHelper.t.sol
↪  b/src/test/forge/BorrowerHelper.t.sol
index f307d11..e4777a0 100644
--- a/src/test/forge/BorrowerHelper.t.sol
+++ b/src/test/forge/BorrowerHelper.t.sol
@@ -148,6 +148,38 @@ contract BorrowerHelperTest is Test {
         borrowerHelper.adjustTrove(address(iclVault), 0.005 ether, collTopUp,
         ↪  collWithdrawal, lusdChange, isDebtIncrease, address(0), address(0));
     }*/

+    function testPOCMedium02AdjustTrove(
+        uint targetCR,
+        uint newTargetCR,
+        bool isDebtIncrease
+    ) public {
+        uint MCR = collateralConfig.getCollateralMCR(address(iclVault));
+        targetCR = bound(targetCR, MCR, 5 ether);
+        newTargetCR = bound(newTargetCR, MCR, 5 ether);
+        uint256 collAmount = 1000 ether;
+
+        uint collTopUp = 0; // @POC: 0 deposits are not supported in
↪  `ReaperVaultV2`
+        uint collWithdrawal = 0;
+        uint lusdChange = 10e18;
+        bool isDebtIncrease = true;
+
+
+        deal(address(icl), address(this), collAmount);
+        icl.approve(address(borrowerHelper), collAmount);
+
+        uint debt = collAmount * 1e18 / targetCR -
↪  borrowerOperations.LUSD_GAS_COMPENSATION();
+        debt -= troveManager.getBorrowingFeeWithDecay(debt);
+        borrowerHelper.openTrove(address(iclVault), collAmount, 0.005 ether,
↪  debt, address(0), address(0));
+
+
+        deal(address(icl), address(this), collTopUp);
+        icl.approve(address(borrowerHelper), collTopUp);
```

```
+
+         vm.expectRevert("Invalid amount"); // @POC: `ReaperVault` reverts on zero
↪   deposit
+         borrowerHelper.adjustTrove(address(iclVault), 0.005 ether, collTopUp,
↪   collWithdrawal, lusdChange, isDebtIncrease, address(0), address(0));
+     }
+
+
      /*function testClaimCollateral() public {
          vm.prank(whale);
          borrowerOperations.withdrawColl(address(iclVault), 1e26 - 130 ether,
          ↪  address(0), address(0));
```

MEDIUM-03 - Proof of Concept

The following patch imports a Foundry unit test to show the issue. Execute `forge test --mt test_POC -vvv`.

```
diff --git a/src/test/forge/BorrowerHelper.t.sol
↪  b/src/test/forge/BorrowerHelper.t.sol
index f307d11..be29cd2 100644
--- a/src/test/forge/BorrowerHelper.t.sol
+++ b/src/test/forge/BorrowerHelper.t.sol
@@ -148,6 +148,38 @@ contract BorrowerHelperTest is Test {
         borrowerHelper.adjustTrove(address(iclVault), 0.005 ether, collTopUp,
         ↪  collWithdrawal, lusdChange, isDebtIncrease, address(0), address(0));
     }*/

+    function testPOCMedium03AdjustTrove(
+        uint targetCR,
+        uint newTargetCR,
+        bool isDebtIncrease
+    ) public {
+        uint MCR = collateralConfig.getCollateralMCR(address(iclVault));
+        targetCR = bound(targetCR, MCR, 5 ether);
+        newTargetCR = bound(newTargetCR, MCR, 5 ether);
+        uint256 collAmount = 1000 ether;
+
+        uint collTopUp = 10e18;
+        uint collWithdrawal = 0;
+        uint lusdChange = 0;
+        bool isDebtIncrease = false; // @POC: `BorrowerHelper` doesn't deposit
↪  collateral if no debt increase
+
+
+        deal(address(icl), address(this), collAmount);
+        icl.approve(address(borrowerHelper), collAmount);
+
+        uint debt = collAmount * 1e18 / targetCR -
↪  borrowerOperations.LUSD_GAS_COMPENSATION();
+        debt -= troveManager.getBorrowingFeeWithDecay(debt);
+        borrowerHelper.openTrove(address(iclVault), collAmount, 0.005 ether,
↪  debt, address(0), address(0));
+
+
+        deal(address(icl), address(this), collTopUp);
+        icl.approve(address(borrowerHelper), collTopUp);
+
+        vm.expectRevert("BorrowerOperations: Insufficient user collateral
↪  balance"); // @POC: `BorrowerOperations` reverts because `BorrowerHelper`
↪  didn't deposit user's assets
```

```
+         borrowerHelper.adjustTrove(address(iclVault), 0.005 ether, collTopUp,
 ↪  collWithdrawal, lusdChange, isDebtIncrease, address(0), address(0));
+     }
+
+
     /*function testClaimCollateral() public {
         vm.prank(whale);
         borrowerOperations.withdrawColl(address(iclVault), 1e26 - 130 ether,
          ↪  address(0), address(0));
```

MEDIUM-04 - Proof of Concept

The following patch imports a Foundry unit test to show the issue. Execute `forge test --mt test_POC -vvv`.

Note: For this PoC to work, another issue needed to be fixed (`HIGH-01`). The patch given to import the PoC modifies the codebase to fix this issue.

```
diff --git a/src/contracts/BorrowerOperations.sol
↪ b/src/contracts/BorrowerOperations.sol
index e86a38c..01b5bac 100644
--- a/src/contracts/BorrowerOperations.sol
+++ b/src/contracts/BorrowerOperations.sol
@@ -505,7 +505,7 @@ contract BorrowerOperations is LiquityBase, Ownable,
↪ CheckContract, IBorrowerOpe
          _requireTroveisActive(contractsCache.troveManager, params._borrower,
          ↪ params._collateral);

          // Confirm the operation is a borrower adjusting their own trove
          ↪ (possibly through the Leverager)
-         assert(msg.sender == params._borrower || msg.sender == leveragerAddress);
+         assert(msg.sender == params._borrower || msg.sender == leveragerAddress
↪ || msg.sender == helperAddress); // @POC: Fix HIGH-01

          contractsCache.troveManager.applyPendingRewards(params._borrower,
          ↪ params._collateral);

diff --git a/src/test/forge/BorrowerHelper.t.sol
↪ b/src/test/forge/BorrowerHelper.t.sol
index f307d11..fc1fe65 100644
--- a/src/test/forge/BorrowerHelper.t.sol
+++ b/src/test/forge/BorrowerHelper.t.sol
@@ -122,6 +122,36 @@ contract BorrowerHelperTest is Test {
          assertEq(iclVault.balanceOf(address(this)), collAmount);
      }

+     function testPOCMedium04AdjustTrove(
+         uint newTargetCR,
+         bool isDebtIncrease
+     ) public {
+         uint MCR = collateralConfig.getCollateralMCR(address(iclVault));
+         uint targetCR = 5 ether;
+         newTargetCR = bound(newTargetCR, MCR, 5 ether);
+         uint256 collAmount = 10000 ether;
+
+         uint collTopUp = 0;
+         uint collWithdrawal = 0;
+        uint lusdChange = 10e18;
+        bool isDebtIncrease = false; // @POC: This indicates LUSD repayment
+
+
+        deal(address(icl), address(this), collAmount);
+        icl.approve(address(borrowerHelper), collAmount);
+
+        uint debt = collAmount * 1e17 / targetCR -
↪   borrowerOperations.LUSD_GAS_COMPENSATION();
+        debt -= troveManager.getBorrowingFeeWithDecay(debt);
+        borrowerHelper.openTrove(address(iclVault), collAmount, 0.005 ether,
↪   debt, address(0), address(0));
+
+
+        deal(address(icl), address(this), collTopUp);
+        icl.approve(address(borrowerHelper), collTopUp);
+
+        vm.expectRevert("BorrowerOps: Caller doesnt have enough LUSD to make
↪   repayment"); // @POC: reverts as `BorrowerHelper` doesn't support LUSD debt
↪   repayment
+        borrowerHelper.adjustTrove(address(iclVault), 0.005 ether, collTopUp,
↪   collWithdrawal, lusdChange, isDebtIncrease, address(0), address(0));
+    }
+
    /*function testAdjustTrove(
        uint collAmount,
        uint targetCR,
```