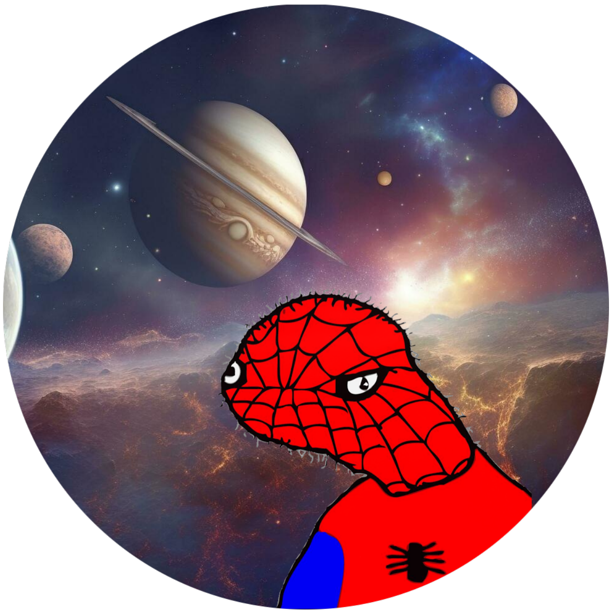

Granary Finance - Leverager Audit Report

Version 1.1

Zigtur

June 11, 2024

Granary Finance - Leverager - Audit Report

Zigtur

June 11, 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 Granary Finance & Leverager

Granary Finance is a decentralized, user-driven borrowing and lending liquidity market inspired by Aave.

The `Leverager` contract integrates with Granary Finance for a user to leverage and deleverage positions.

Security Assessment Summary

***Review commit hash* -** 6906bef91aa330dc5d248ff4101fc9846fad1ecd

***Fixes review commit hash* -** e7828476468aa7eba490fc03b5f735d9e25db518

Deployment chains

- All EVM chains/rollups

Scope

The following smart contracts are in scope of the review:

- contracts/leverager/Leverager.sol

Risk Classification

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

Note: Informational findings may not raise security concerns but are notable and require developers' attention.

Issues

HIGH-01 - Deleveraging can result in less healthy positions

Description:

The deleveraging process withdraws all liquidity deposited by the user in Granary's lending pool for the given asset.

When a user borrows additional assets and then deleverages their position, withdrawing asset liquidity can make their position less healthy than before deleveraging.

The user will not be able to ensure a minimal health factor due to a lack of health factor check.

Impact:

Users can't control their position's health during deleveraging.

Code snippet:

The `_deleverage` function lacks a check to ensure a minimum health factor on the user's position.

```
    function _deleverage(
        address _asset
    ) internal {
        // ...

        lendingPool.flashLoan(
            address(this),
            assets_, // [_asset]
            amounts_,// [_borrowAmount]
            modes_,  // [0] no debt
            msg.sender, // onBehalfOf
            abi.encode(false, msg.sender, 0),
            0 // referral code
        );

        // @POC: Lack of Health Factor check
    }
```

For comparison, the `_leverage` function implements an health factor check.

```
    function _leverage(
        address _asset,
        uint256 _initialDeposit,
        uint256 _borrowAmount,
        uint256 _minHealthFactor
    ) internal {
        // ...

        lendingPool.flashLoan(
            address(this),
            assets_, // [_asset]
            amounts_,// [_borrowAmount]
            modes_,  // [2] variable debt
            msg.sender, // onBehalfOf
            abi.encode(true, msg.sender, _initialDeposit),
            0 // referral code
        );

        // @POC: Check Health Factor
        (,,,,, uint256 healthFactor_) = lendingPool.getUserAccountData(
           msg.sender);
        if (healthFactor_ < _minHealthFactor) revert
           Leverager__INVALID_HEALTH_FACTOR();
    }
```

Recommendation:

Add an health factor check after deleveraging to let the user ensures his position is healthy enough.

A patch available in Appendix implements this recommendation.

Resolution

Fixed. The recommendation has been followed by applying the patch.

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

The Solidity compiler version used for `Leverager` is `0.8.23`. This version includes the `PUSH0` instruction.

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

Consider compiling `Leverager` with Solidity `0.8.19` to remove the `PUSH0` instruction.

Resolution

Fixed. The Solidity compiler is now version `0.8.19`.

INFO-01 - Users can't get the paused status

The `paused` variable indicates weither the `Leverager` is paused or not. This variable being `internal`, users can't get the pausing status of the contract.

```
    bool internal paused;
```

Consider making the `paused` variable **`public`** to allow retrieving the pausing status off-chain.

Resolution

Fixed. The `paused` variable is now **`public`**.

INFO-02 - Native support can't be activated after deployment

The support for native leverage and deleverage is defined during deployment. Once deployed, this configuration can't be modified.

Resolution

Acknowledged. The team states:

> Since this contract acts like a library, we can just redeploy it if we want to enable native.

INFO-03 - 1-Click-Looping codebase doesn't include test

The `Leverager` contract doesn't include a test suite. Functional and unit tests are a good practice for security purposes.

Resolution

Fixed. The functions without access control are tested.

INFO-04 - Hardhat can't compile `Leverager`

The `Leverager` contract requires Solidity `0.8.23`.

However, this version is not configured in `hardhat.config.ts`.

```
  solidity: {
    compilers: [
      {
        version: "0.6.12",
        settings: {
          optimizer: { enabled: true, runs: 200 },
        },
      },
      {
        version: "0.7.5",
        settings: {
          optimizer: { enabled: true, runs: 200 },
        },
      },
      {
        version: "0.6.6",
        settings: {
          optimizer: { enabled: true, runs: 200 },
        },
      },
    ],
  },
```

Resolution

Fixed. The Solidity compiler version `0.8.19` is now configured in Hardhat.

Appendix

HIGH-01 - Fix patch

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

```
diff --git a/contracts/leverager/Leverager.sol b/contracts/leverager/Leverager.sol
index 6bca514..5fa4bd7 100644
--- a/contracts/leverager/Leverager.sol
+++ b/contracts/leverager/Leverager.sol
@@ -59,18 +59,19 @@ contract Leverager is IFlashLoanReceiver, AccessControl {
         _leverage(_asset, _initialDeposit, _borrowAmount, _minHealthFactor);
     }

-    function deleverageNative() external {
+    function deleverageNative(uint256 _minHealthFactor) external {
         if (address(weth) == address(0)) revert Leverager__NATIVE_LEVERAGE_NOT_ACTIVATED();
-        _deleverage(address(weth));
+        _deleverage(address(weth), _minHealthFactor);
         weth.withdraw(weth.balanceOf(address(this)));
         (bool success_, ) = payable(msg.sender).call{value: address(this).balance}("");
         if (!success_) revert Leverager__TRANSFER_FAILED();
     }

     function deleverageERC20(
-        address _asset
+        address _asset,
+        uint256 _minHealthFactor
     ) external {
-        _deleverage(_asset);
+        _deleverage(_asset, _minHealthFactor);
         uint256 assetBalance_ = IERC20(_asset).balanceOf(address(this));
         if (assetBalance_ != 0)
             IERC20(_asset).safeTransfer(msg.sender, assetBalance_);
@@ -127,10 +128,12 @@ contract Leverager is IFlashLoanReceiver, AccessControl {
      * @param _asset to deleverage
      */
     function _deleverage(
-        address _asset
+        address _asset,
+        uint256 _minHealthFactor
     ) internal {
         if (paused) revert Leverager__PAUSED();
         if (_asset == address(0)) revert Leverager__INVALID_INPUT();
+        if (_minHealthFactor < MIN_HF) revert Leverager__INVALID_HEALTH_FACTOR();

         address debtToken_ = lendingPool.getReserveData(_asset).variableDebtTokenAddress;

@@ -152,6 +155,9 @@ contract Leverager is IFlashLoanReceiver, AccessControl {
             abi.encode(false, msg.sender, 0),
             0 // referral code
         );
+
+        (,,,,, uint256 healthFactor_) = lendingPool.getUserAccountData(msg.sender);
+        if (healthFactor_ < _minHealthFactor) revert Leverager__INVALID_HEALTH_FACTOR();
     }

     function executeOperation(
```