Code Security Audit Report

For

Blend

29th Oct 2024

Damocles

Table of Contents

Damocles

Project Executive Summary

TYPES	MAINNET	AUDIT METHOD
Lend Protocol	EDU Chain	Cross-manual review; Static Analysis
DEVELOP LAUNGE	**CODEBASE**	**Commit ID**
Solidity	Contract Link	664a8be5

VVulnerability Summary

2	2	0	0
Total Findings	Solved	Pending	Acknowledged

SEVERITY	STATUS	DESCRIPTION
CRITICAL	0	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risk
HIGH	0	High risks can include centralization issues and logical errors. Under specific circumstances, these high risks can lead to loss of funds and/or control of the project
MEDIUM	1 Solved	Medium risks may not pose a direct risk to user's funds, but they can affect the overall functioning of a platform
LOW	0	Low risks can be any of the above, but on a smaller scale. They generally do not compromise the over all integrity of the project, but they may be less efficient than other solution

INFO	**1 Solved**	Info errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code

Audit Scope Summary

Code Repository: https://github.com/Blend-Blend/blend-contract	
File Name	**Commit ID**
Dependencies/*.sol	664a8be506142e21ac8e318db2083a5ffd40d98f
Deployments/*sol	664a8be506142e21ac8e318db2083a5ffd40d98f
FlashLoan/*.sol	664a8be506142e21ac8e318db2083a5ffd40d98f
Interface/*.sol	664a8be506142e21ac8e318db2083a5ffd40d98f
Misc/*.sol	664a8be506142e21ac8e318db2083a5ffd40d98f
Periphery/*sol	664a8be506142e21ac8e318db2083a5ffd40d98f
Protocol/*sol	664a8be506142e21ac8e318db2083a5ffd40d98f

Audit Approach Summary

This report has been prepared **ThrustPad** to discover issues and vulnerabilities in the source code of the **Thrustpad Main Repo** as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Cross Manual Review and Static Analysis techniques

- The auditing process pays special attention to the following considerations:
- Testing the smart contracts against both common and uncommon attack vectors.

- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- .Thorough line-by-line manual review of the entire codebase by industry experts
- Cross-audit mode of the current code by more than three security engineers.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live

Damocles

Audit Result

BLD-01(Medium):The incorrect token authorization can lead to unverified over-collateralized withdrawal even

Category	Severity	Location	Status
Logic Issue	Medium	ParaSwapRepayAdapter.sol:182	Solved

Description

Due to the existence of slippage in on-chain transactions, a portion of assets may remain on-chain when users conduct transactions. These assets can be maliciously extracted when interacting with external contracts.

Vulnerability Analysis

1. In the '***swapandRepay***' function, the '***_buyonparaswap***' function is called, which sets the ***'assettoswap'*** value of the ***tokenTransferProxy*** to the ***collateralAmount*** passed in..

```
function _swapAndRepay(
  bytes calldata params,
  uint256 premium,
  address initiator,
  IERC20Detailed collateralAsset,
  uint256 collateralAmount
) private {
  (
    IERC20Detailed debtAsset,
    uint256 debtRepayAmount,
    uint256 buyAllBalanceOffset,
    uint256 rateMode,
    bytes memory paraswapData,
    PermitSignature memory permitSignature
  ) = abi.decode(params, (IERC20Detailed, uint256, uint256, uint256, bytes, PermitSignature));

  debtRepayAmount = getDebtRepayAmount(
    debtAsset,
    rateMode,
    buyAllBalanceOffset,
    debtRepayAmount,
    initiator
  );

  uint256 amountSold = _buyOnParaSwap(
    buyAllBalanceOffset,
    paraswapData,
    collateralAsset,
    debtAsset,
    collateralAmount,
    debtRepayAmount
  );
```

```
  uint256 maxAmountToSwap,
  uint256 amountToReceive
) internal returns (uint256 amountSold) {
  (bytes memory buyCalldata, IParaSwapAugustus augustus) = abi.decode(
    paraswapData,
    (bytes, IParaSwapAugustus)
  );

  require(AUGUSTUS_REGISTRY.isValidAugustus(address(augustus)), 'INVALID_AUGUSTUS');

  {
    uint256 fromAssetDecimals = _getDecimals(assetToSwapFrom);
    uint256 toAssetDecimals = _getDecimals(assetToSwapTo);

    uint256 fromAssetPrice = _getPrice(address(assetToSwapFrom));
    uint256 toAssetPrice = _getPrice(address(assetToSwapTo));

    uint256 expectedMaxAmountToSwap = amountToReceive
      .mul(toAssetPrice.mul(10**fromAssetDecimals))
      .div(fromAssetPrice.mul(10**toAssetDecimals))
      .percentMul(PercentageMath.PERCENTAGE_FACTOR.add(MAX_SLIPPAGE_PERCENT));

    require(maxAmountToSwap <= expectedMaxAmountToSwap, 'maxAmountToSwap exceed max slippage');
  }

  uint256 balanceBeforeAssetFrom = assetToSwapFrom.balanceOf(address(this));
  require(balanceBeforeAssetFrom >= maxAmountToSwap, 'INSUFFICIENT_BALANCE_BEFORE_SWAP');
  uint256 balanceBeforeAssetTo = assetToSwapTo.balanceOf(address(this));

  address tokenTransferProxy = augustus.getTokenTransferProxy();
  assetToSwapFrom.approve(tokenTransferProxy, 0);
  assetToSwapFrom.approve(tokenTransferProxy, maxAmountToSwap);

  if (toAmountOffset != 0) {
    // Ensure 256 bit (32 bytes) toAmountOffset value is within bounds of the
```

2. At the same time, since the final execution is done through an external contract call

   (bool success,) = address(augustus).call(buyCalldata), it provides users with the opportunity to interact externally.

```
}
(bool success, ) = address(augustus).call(buyCalldata);
if (!success) {
  // Copy revert reason from call
  assembly {
    returndatacopy(0, 0, returndatasize())
    revert(0, returndatasize())
  }
}
```

3. The execution parameters of **buyCalldata** are mainly decoded from **paraswapData** without any verification, allowing external contracts to potentially bypass the exchange and return a success status by constructing data in a certain way.

```
    uint256 toAmountOffset,
    bytes memory paraswapData,
    IERC20Detailed assetToSwapFrom,
    IERC20Detailed assetToSwapTo,
    uint256 maxAmountToSwap,
    uint256 amountToReceive
  ) internal returns (uint256 amountSold) {
    (bytes memory buyCalldata, IParaSwapAugustus augustus) = abi.decode(
      paraswapData,
      (bytes, IParaSwapAugustus)
    );
```

Recommendation

1. **The logical solution suggestion** is to revoke authorization after each repayment or to verify the paraswapData parameters. **As for the dependency solution**, upgrading Periphery to the paraswap-excess branch can help enhance stability.

LPB-02(Info): Hardcoded address configuration error.

Category	Severity	Location	Status
Code Issue	Medium	UiPoolDataProviderV3.sol:29 PoolDataProvider.sol:25	Solved

Description

When using Fork code, failing to modify hardcoded addresses in the original code can lead to unexpected errors.

Vulnerability Analysis

1. The ETH and MKR addresses in the **'Blend\contracts\misc\PoolDataProvider.sol'** path are both original addresses on ETH mainnet. It is important to pay close attention to address configuration when deploying on the mainnet.

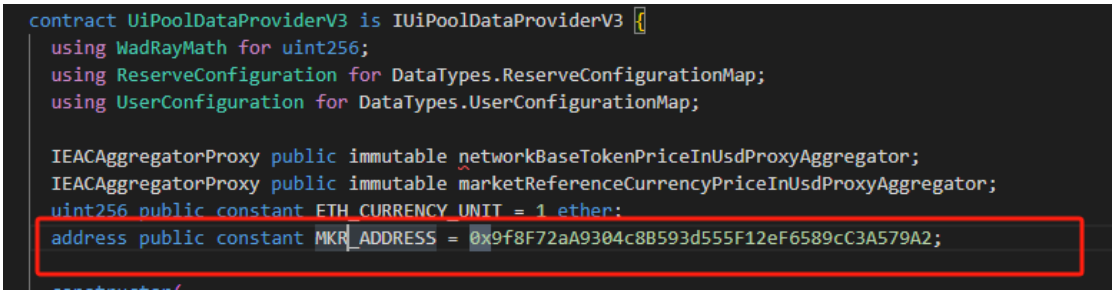
```
*/
contract PoolDataProvider is IPoolDataProvider {
  using ReserveConfiguration for DataTypes.ReserveConfigurationMap;
  using UserConfiguration for DataTypes.UserConfigurationMap;
  using WadRayMath for uint256;

  address constant MKR = 0x9f8F72aA9304c8B593d555F12eF6589cC3A579A2;
  address constant ETH = 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE;

  /// @inheritdoc IPoolDataProvider
  IPoolAddressesProvider public immutable ADDRESSES_PROVIDER;
```

2. In the Blend\contracts\periphery\misc\UiPoolDataProviderV3.sol file, there is also an issue with using original file addresses configuration. It is important to pay attention to this when

deploying on the mainnet.



```
contract UiPoolDataProviderV3 is IUiPoolDataProviderV3 {
  using WadRayMath for uint256;
  using ReserveConfiguration for DataTypes.ReserveConfigurationMap;
  using UserConfiguration for DataTypes.UserConfigurationMap;

  IEACAggregatorProxy public immutable networkBaseTokenPriceInUsdProxyAggregator;
  IEACAggregatorProxy public immutable marketReferenceCurrencyPriceInUsdProxyAggregator;
  uint256 public constant ETH_CURRENCY_UNIT = 1 ether;
  address public constant MKR_ADDRESS = 0x9f8F72aA9304c8B593d555F12eF6589cC3A579A2;
```

Recommendation

Create a dedicated configuration file like Config.json to centrally manage and store the addresses that need to be configured when deploying on the mainnet, and conduct rigorous functional testing after going live on the mainnet.

APPENDIX

Audit Categories

Categories	Description
Code Issues	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities
Logic Issues	Logic Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralizedcontrols over the code

About

Damocles is a 2023 web3 security company specializing in online security services, including smart contract audit, Product audit, penetration testing, GameFi security audit and cheat detection.

Main Web: https://damocleslabs.com/

Twitter: https://twitter.com/DamoclesLabs

Email: support@damocleslabs.com