Cod3x - Lend

Audit Report

Version 1.2

Zigtur

October 29, 2024

Cod3x - Lend - Audit Report

Zigtur

October 29, 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 Cod3x Lend

Cod3x Lend is a cryptocurrency-native money market module implemented in Cod3x Modular Reserve Protocols on the Ethereum Virtual Machine. Compared to existing money markets, Cod3x Lend enhances functionality by introducing MiniPools, Collateral Flow Controls, and collateral Rehypothecation.

Cod3x Lend aims to address the industry's most pressing issues: liquidity fragmentation and risk management.

Security Assessment Summary

Review commit hash - 689a3ad

Fixes review commit hash - XXXX

Deployment chains

- All EVM chains/rollups

Scope

The following smart contracts are in scope of the review:

- contracts/protocol/*

A specific commit 866da3d has been set for the contracts under `interestRateStrategies`.

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 - aToken share price is under-valued

Description

Scope:

- Oracle.sol#L118-L120

The lending pool aToken has a 1:1 rate with the underlying token. This means that the price of an underlying token corresponds to the aToken price.

The `Oracle` contract transform this aToken price (representing **1 aToken**) through a call to `convertIntoShares`. This conversion is incorrect and will not result in a price for **1 share**.

```
// if `asset` is an aToken then convert the price from asset to share.
if (asset != underlying) {
    return ATokenNonRebasing(asset).convertToShares(finalPrice);
} else {
```

Proof of Concept

Let's say the conversion rate between `aToken` and `share` is `1.25` (also known as the normalized income):

- `finalPrice = 1e18`
- `convertToShares(1e18) = 1e18 / 1.25 = 0.8`
- `sharePrice = 0.8` => This is incorrect, 0.8 is the amount of share for **1 aToken** but we want the amount of **aToken** for **1 share**.

The correct calculations is:

- `amountScale = 1e18`
- `one_share = 1e18`
- `finalPrice = 1e18`
- `convertToAssets(one_share) = 1e18 * 1.25 = 1.25e18` => This is the amount of aToken per share
- `sharePrice = 1.25e18 * finalPrice / amoutScale = 1.25e18` => This gives the price of 1 share.

This calculation can be simplified into `convertToAssets(one_share * finalPrice / amountScale)` => `convertToAssets(finalPrice)`.

An interesting edge-case is an asset that does not use the same scaling than the price. For example USDC uses 6 decimals.

- `amountScale = 1e6`
- `one_share = 1e6`
- `finalPrice = 1e18`
- `convertToAssets(one_share) = 1e6 * 1.25 = 1.25e6` => This is the amount of aToken per share
- `sharePrice = 1.25e6 * finalPrice / amoutScale = 1.25e18` => This gives the price of 1 share on a 18 decimals scale.

Then, the provided solution works with non 18 decimals tokens.

Recommendation

The share price calculations in `Oracle.getAssetPrice` function must be modified to correctly calculate the price per share.

```
// if `asset` is an aToken then convert the price from asset to share.
if (asset != underlying) {
    return ATokenNonRebasing(asset).convertToAssets(finalPrice);
} else {
```

A patch is given in Appendix to apply this recommendation.

Resolution

Cod3x team: Fixed in PR14 commit f01f043.

Zigtur: Fixed. Recommended patch has been applied. `convertToAssets` is now used.

HIGH-02 - Minipool reserve initialization logic is incorrect

Description

- MiniPoolReserveLogic.sol#L177-L180
- MiniPoolConfigurator.sol#L77-L101

The MiniPool reserve initialization logic does not work as expected. It implements an incorrect check due to **two different issues**. This function is used as part of the reserve initialization executed from the `MiniPoolConfigurator`.

```
function init(
    DataTypes.MiniPoolReserveData storage reserve,
    IAERC6909 aTokenAddress,
    ...
) internal {
    require(
        aTokenAddress.getUnderlyingAsset(reserve.aTokenID) == address(0), // @POC:
        ↪  incorrect check
        Errors.RL_RESERVE_ALREADY_INITIALIZED
    );
```

The `MiniPoolReserveLogic.init` function tries to ensure that the `aToken` associated to the `reserve.aTokenID` is zero. However, `reserve.aTokenID` is read from **storage** which **has not been initialized yet**. This value will always be zero, leading the `MiniPool` to be unable to initialize new reserves when the zero token ID is used.

Moreover, fixing this storage issue will have no impact as this code is also prone to another issue. Part of the workflow between `MiniPoolConfigurator`, `MiniPool` and `ATokenERC6909` is shown in the following diagram.

The workflow shows that `ATokenERC6909._underlyingAssetAddresses` is set in the `_initializeATokenID` function. However, the `MiniPoolReserveLogic.init` function reads this value and expect it to be `address(0)`. This will never be the case, which will lead to denial of service.

Recommendation

The `MiniPoolReserveLogic.init` logic must be reworked to ensure that the underlying asset of the aTokenID is the current asset. The `MiniPool` can't directly check that the aTokenId wasn't used before.

A patch is given in Appendix to apply this recommendation.

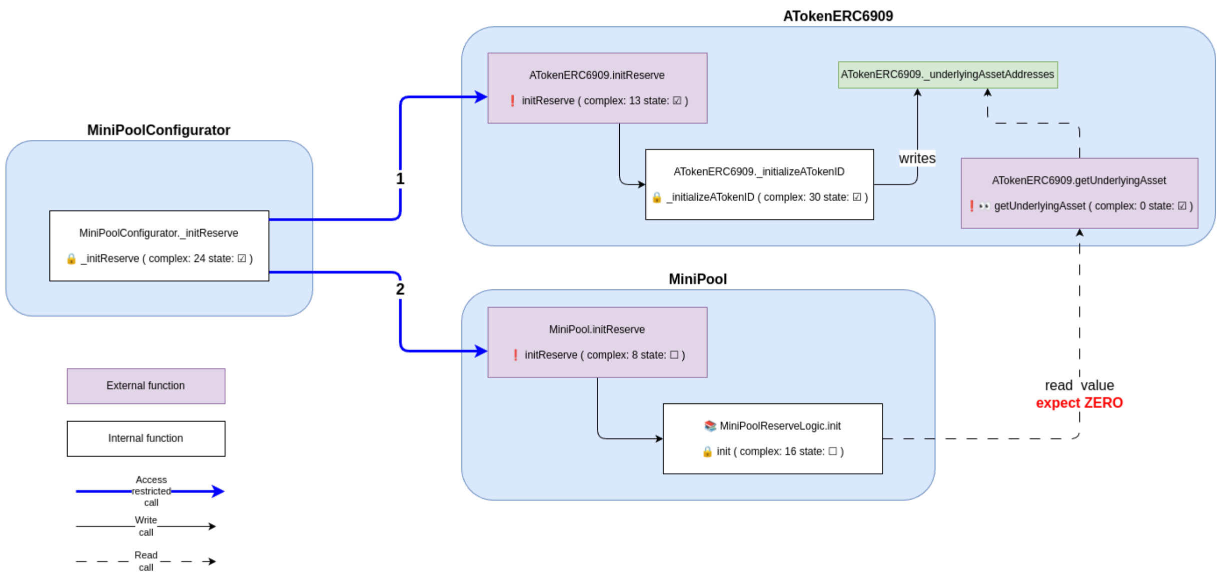
ATokenERC6909
ATokenERC6909.initReserve
❗ initReserve (complex: 13 state: ☑)
ATokenERC6909._underlyingAssetAddresses
ATokenERC6909._initializeATokenID
🔒 _initializeATokenID (complex: 30 state: ☑)
writes
ATokenERC6909.getUnderlyingAsset
❗ 👀 getUnderlyingAsset (complex: 0 state: ☑)
MiniPoolConfigurator
MiniPoolConfigurator._initReserve
🔒 _initReserve (complex: 24 state: ☑)
1
2
MiniPool
MiniPool.initReserve
❗ initReserve (complex: 8 state: ☐)
📚 MiniPoolReserveLogic.init
🔒 init (complex: 16 state: ☐)
read value
expect ZERO
External function
Internal function
Access restricted call
Write call
Read call

Figure 1: HIGH-02 - Reserve initialization workflow

Resolution

Cod3x team: Fixed in PR14 commit c2c27e0.

Zigtur: Fixed. Recommended patch has been applied.

HIGH-03 - A user can use `repayWithAtokens` on behalf of anyone

Description

Scope:

- LendingPool.sol#L215-L226
- BorrowLogic.sol#L349-L384

The new `repayWithAtokens` function allows repaying debt through the use of the `aToken` balance of an account.

However, this new function does not ensure that the `msg.sender` is allowed to use this function on behalf of the `onBehalfOf` address.

Code snippet

The `BorrowLogic.repayWithAtokens` function doesn't ensure that `msg.sender` is allowed by `params.onBehalfOf`.

```
function repayWithAtokens(
    repayParams memory params,
    mapping(address => mapping(bool => DataTypes.ReserveData)) storage _reserves,
    mapping(address => DataTypes.UserConfigurationMap) storage _usersConfig
) internal returns (uint256) {
    // ...

    IVariableDebtToken(reserve.variableDebtTokenAddress).burn(
        params.onBehalfOf, paybackAmount, reserve.variableBorrowIndex
    );

    // ...

    IAToken(aToken).burn(params.onBehalfOf, aToken, paybackAmount,
    ↪   reserve.liquidityIndex); // @POC: burn aToken without approval

    emit Repay(params.asset, params.onBehalfOf, msg.sender, paybackAmount);

    // ...
}
```

Note that a check is done in `ValidationLogic.validateRepay`, but this can be easily bypassed through providing `amountSent != type(uint256).max`.

Recommendation

The `repayWithAtokens` functionality must only be accessible when the `msg.sender` is approved by or is the `onBehalfOf` address.

Resolution

Cod3x team: Fixed in PR14 commit 84c2e7f.

Zigtur: Fixed. `onBehalfOf` parameter has been removed and `msg.sender` is now used.

HIGH-04 - MiniPool liquidations do not work

Description

Scope:

- MiniPoolCollateralManager.sol#L89-L97
- MiniPoolLiquidationLogic.sol#L26-L34

During a liquidation, a delegatecall to function `MiniPoolCollateralManager.liquidationCall` is made.

However, the parameters encoding for the delegatecall does not comply with the expected parameters.

Code snippet

The `MiniPoolCollateralManager.liquidationCall` function expects 7 input parameters.

```
function liquidationCall(
    address collateralAsset,
    bool,  // @POC: boolean input expected
    address debtAsset,
    bool,  // @POC: boolean input expected
    address user,
    uint256 debtToCover,
    bool receiveAToken
) external override returns (uint256, string memory) {
```

However, `MiniPoolLiquidationLogic.liquidationCall` only encodes 5 parameters.

```
function liquidationCall(liquidationCallParams memory params) external {
    address collateralManager =
    ↪  ILendingPoolAddressesProvider(params._addressesProvider)
        .getLendingPoolCollateralManager();

    //solium-disable-next-line
    (bool success, bytes memory result) = collateralManager.delegatecall(
        abi.encodeWithSignature(
            "liquidationCall(address,bool,address,bool,address,uint256,bool)",
            params.collateralAsset,  // @POC: should be followed by a boolean
             ↪  argument
            params.debtAsset,        // @POC: should be followed by a boolean
             ↪  argument
            params.user,
            params.debtToCover,
            params.receiveAToken
        )
    );
```

Proof of Concept

One of the unit test fails and shows this issue. It can be reproduced through Foundry:

```
forge test --mt testMiniPoolLiquidation -vvv
```

Recommendation

Consider removing the boolean input parameters from the `MiniPoolCollateralManager.liquidationCall` function as they are not used.

If the interface should be kept as is, then add two `false` boolean to the `liquidationCall` delegatecall encoding.

A patch is given in Appendix to encode two `false` boolean.

Resolution

Cod3x team: Fixed in commit 7324bfa.

Zigtur: Fixed.

HIGH-05 - MiniPool's debt in LendingPool will never be entirely repaid when debt increases faster than the aToken interest rate

Description

Scope:

- BorrowLogic.sol#L412-L423
- MiniPool.sol#L192-L214

The `MiniPool` implements a risky mechanism to borrow funds from the `LendingPool`. This mechanism first takes some of a token liquidity. Then, debt is minted with a 1:1 rate to the `MiniPool` when token liquidity is taken. Finally, the `MiniPool` deposits the token liquidity back into the lending pool to get `aToken` which are also minted at a `1:1` rate.

However, during repayment from the `MiniPool`, debt may have increased faster than the `aToken` interested rate. In this case, the `MiniPool` will not be able to repay its debt.

Recommendation

The whole `MiniPool` borrowing into `LendingPool` mechanism must be reviewed. The current mechanism is highly unsafe and may hide other critical issues.

The current implementation makes the assumption that the debt and the collateral will grow at the same rate (see HIGH-06). The modified mechanism must ensure that such assumption is not made as this is incorrect.

Resolution

Cod3x team: Fixed in PR14 commits effd362, 0414e714 and ab7b851. A specific documentation is available on Notion.

Zigtur: Fixed. The Minipool interest rate calculations have been reviewed to depend on the main lending pool interest rate. A 1% safe margin is added on this rate.

This new rate calculation ensures that the Minipool will be able to repay all its debt.

HIGH-06 - AERC6909 tokens will be stuck in MiniPool when aToken increases faster than the debt in LendingPool

Description

Scope:

- BorrowLogic.sol#L412-L423
- MiniPool.sol#L192-L214
- MiniPoolDepositLogic.sol#L91-L95
- MiniPool.sol#L342-L357

When borrowing from the `LendingPool`, the `MiniPool` mints `AERC6909` tokens to itself. This corresponds to the amount of `AToken` borrowed from the main `LendingPool`.

However, when repaying its debt to the `LendingPool`, the `MiniPool` only withdraw the amount of `AERC6909` required to repay the debt.

Then, if the `AToken` increased faster than the debt at the `LendingPool` level, the `MiniPool` will withdraw less `AERC6909` tokens than its balance.

This leads `AERC6909` tokens being locked into the `MiniPool` contract.

Recommendation

The whole `MiniPool` borrowing into `LendingPool` mechanism must be reviewed. The current mechanism is highly unsafe and may hide other critical issues.

The current implementation makes the assumption that the debt and the collateral will grow at the same rate (see HIGH-05). The modified mechanism must ensure that such assumption is not made as this is incorrect.

Resolution

Cod3x team: Fixed in PR14 commits 2ffa3f1 and 79f6e39.

Zigtur: Fixed. The AERC6909 tokens remaining due to the minipool interest rate being higher than the lending pool interest rate will be sent to the treasury.

HIGH-07 - Interest rate calculation will revert when the reserve is heavily used

Description

Scope:

- MiniPoolDefaultReserveInterestRate.sol#L114-L116
- MiniPoolPiReserveInterestRateStrategy.sol#L83-L85

During interest rate calculations, the current flow and limit are used through the following calculation:

```
114 availableLiquidity = IERC20(reserve).balanceOf(aToken)
115     + IAToken(reserve).convertToShares(flowLimiter.getFlowLimit(underlying,
        ↪  minipool))
116     - IAToken(reserve).convertToShares(flowLimiter.currentFlow(underlying,
        ↪  minipool));
```

This calculation will revert when the balance of `aToken` is close to zero. This is due to the `currentFlow` increasing when the flow limit is reached due to interest rate.

Then, this calculation may end up with `flowLimiter.getFlowLimit < currentFlow`, leading to an underflow issue. This will break the whole `MiniPool`.

Recommendation

The interest rate strategies must not expect the `currentFlow` to be lower than the flow limit.

Resolution

Cod3x team: Fixed in PR14 commit e1d5673.

Zigtur: Fixed. `getFlowLimit` now returns the current flow when it is greater than the maximum minipool debt. This ensures that `flowLimiter.getFlowLimit < currentFlow` can't be triggered.

HIGH-08 - Rewards controller incorrectly track the total difference

Description

Scope:

- RewardsController.sol#L84

The `RewardsController` contract aims to accounts for the rewards owed to a user depositing in the main lending pool and in the mini pool. A `_totalDiff` variable is used to reflect how much a specific `aToken` has been deposited into the `aToken6909`.

However, this `_totalDiff` is depending on each asset and should not be an unique value. For example, ETH deposits could impact USDC rewards and vice-versa.

Recommendation

The `_totalDiff` values must be dependent on the asset.

Resolution

Cod3x team: Fixed in PR14 commit 4e4a87a.

Zigtur: Fixed. The `_totalDiff` is now a mapping. Diff values are asset dependent.

MEDIUM-01 - `_rebalance` will be unusable for a limited period after the first rehypothecation

Description

Scope:

- AToken.sol#L527
- AToken.sol#L529-L530
- AToken.sol#L535

`_rebalance` calculates the current profit from rehypothecation by interacting with the vault. However, this calculation leads to a denial of service when the current profit is negative (revert due to integer underflow).

The current profit is calculated through the following code:

```
function _rebalance(uint256 _amountToWithdraw) internal {
    // ...

    // how much has been allocated as per our internal records?
    uint256 currentAllocated = farmingBal;
    // what is the present value of our shares?
    uint256 ownedShares = IERC20(address(vault)).balanceOf(address(this));
    uint256 sharesToAssets = vault.convertToAssets(ownedShares);
    uint256 profit;
    // ...

    // if we have profit that's more than the threshold, record it for withdrawal
     ↪  and redistribution
    if (sharesToAssets - currentAllocated >= claimingThreshold) {
        profit = sharesToAssets - currentAllocated;
    }
```

However, most ERC-4626 vault implementations are rounding down when calculating `convertToAssets` and `convertToShares`. This double rounding down leads to `sharesToAssets` being less than `currentAllocated`.

Scenario

First deposit of `1000` tokens:

- `currentAllocated = farmingBal = 0`
- `vault.deposit(1000, this)` is called

    - `deposit` turns assets into shares by rounding down
    - `999` shares are minted

- `farmingBal += 1000`

Second deposit of `1000` tokens:

- `currentAllocated = farmingBal = 1000`
- `sharesToAssets` is calculated
    - `vault.convertToAssets(999)` will also round down
    - `sharesToAssets = 999`
- `sharesToAssets - currentAllocated` is executed
    - `999 - 1000` => integer underflow => **revert**

Recommendation

The `sharesToAssets - currentAllocated` must be executed only if `sharesToAssets > currentAllocated`.

Resolution

Cod3x team: Fixed in PR14 commit bfa4a76.

Zigtur: Fixed. Recommendation has been applied. The subtraction is not executed when `sharesToAssets > currentAllocated`.

MEDIUM-02 - Oracle uses the deprecated `latestAnswer` function from Chainlink

Description

Scope:

- Oracle.sol#L110-L113

The `Oracle` contract retrieves price answer from Chainlink through `latestAnswer`. This function is marked as deprecated according to Chainlink documentation.

Recommendation

The `latestRoundData` function must be used. Moreover, when fetching a price from a Chainlink oracle, consider the following checks:

- Consider basic sanity checks on the `answer` and the `updatedAt` values

```
require(answer > 0);
require(updatedAt != 0);
require(updatedAt <= block.timestamp);
```

- Consider a check for staleness with a reasonable `timeout`

```
require(block.timestamp - updatedAt < timeout);
```

The timeout can be configured through a setter only allowed to the owner.

- Consider checking the oracle price range (`minAnswer` and `maxAnswer`)

Resolution

Cod3x team: Fixed in PR14 commits bebc6e6 and ebe604d.

Zigtur: Fixed. `latestRoundData` is called to retrieve the price. Multiple checks are implemented (timeouts, prices, …).

MEDIUM-03 - AERC6909 reserve initialization can override existing tokens

Description

- ATokenERC6909.sol#L103-L123

Reserve initialization in `ATokenERC6909` doesn't ensure that the `aTokenID` and the `debtTokenId` were not already used.

As these parameters are derived from the main lending pool reserve identifiers, this lack of check can lead to the reinitialization of existing and already supported tokens.

Code snippet

The `ATokenERC6909.initReserve` function does not check that the `aTokenID` and `debtTokenID` are not already used.

```
function initReserve(
    ...
) external returns (uint256 aTokenID, uint256 debtTokenID, bool isTrancheRet) {
    require(
        msg.sender == address(_addressesProvider.getMiniPoolConfigurator()),
        Errors.LP_CALLER_NOT_LENDING_POOL_CONFIGURATOR
    );
    (aTokenID, debtTokenID, isTrancheRet) = getIdForUnderlying(underlyingAsset);
    ↪  // @POC: get IDs
    if (isTrancheRet) {
        _totalTrancheTokens++;
        _isTranche[aTokenID] = true;
        _isTranche[debtTokenID] = true;
    } else {
        _totalUniqueTokens++;
    }
    _initializeATokenID(aTokenID, underlyingAsset, name, symbol, decimals); //
    ↪  @POC: use ID
    _initializeDebtTokenID(debtTokenID, underlyingAsset, name, symbol, decimals);
    ↪  // @POC: use ID
}
```

Recommendation

`ATokenERC6909.initReserve` function must check that the `aTokendID` and `debtTokenID` are not already used. Checking only one of these two is enough, as both are linked together (`aTokendID + 1000 == debtTokenID`).

A patch is given in Appendix to apply this recommendation.

Resolution

Cod3x team: Fixed in PR14 commit 55971f0.

Zigtur: Fixed. The `getIdForUnderlying` function doesn't allow overwriting and retrieves the data from the minipool when needed.

MEDIUM-04 - User can’t control their expected LTV threshold during borrow

Description

Scope:

- BorrowLogic.sol#L135-L141
- WithdrawLogic.sol#L152
- MiniPoolBorrowLogic.sol#L123-L129
- MiniPoolWithdrawLogic.sol#L148

A new feature has been added. The LTV and liquidation thresholds now depends on the volatility of the user’s assets. For example, a user borrowing ETH with USDC as only collateral would have a 90% liquidation threshold while a user borrowing ETH with USDC and PEPE (volatile asset) would have a 70% liquidation threshold.

However, this new feature opens a new attack vector.

A user is able to transfer collateral tokens (namely `AToken` for lending pool and `ATokenERC6909` for minipool) to any other user. Then, a malicious user can make any borrow transaction expecting low or medium volatility and send a high volatile asset. This will make the borrow operation fail.

Scenario

Bob wants to borrow ETH by depositing USDC with a LTV of 75%. The LTV and liquidation threshold for this low volatility asset are 85% and 90%. He deposits USDC as collateral.

Alice deposits 0.1 PEPE and gets 0.1 aPEPE. She transfers 0.1 aPEPE to Bob.

Bob tries to borrow with a LTV of 75%, his borrow operation fails due to the PEPE collateral lowering the LTV threshold to 70%.

Recommendation

Due to the new feature, the user should be able to choose which asset is used as collateral or not. This will fix this issue as it would require the user to manually set the collateral as allowed.

Resolution

Cod3x team: Fixed in PR14 commit 1cc00ab.

Zigtur: Fixed. The functionality has been removed.

MEDIUM-05 - Rehypothecation vault can't be modified once set

Description

Scope:

- AToken.sol#L597-L600

The `AToken.setVault` function checks that the current `vault` address is `address(0)` and that the newly configured `_vault` supports the correct underlying asset.

This leads the `vault` address to be settable only once by the admin. Once it is set, the vault can never be modified. Funds can still be withdrawn from it by setting a farming percentage of `0` and a farming drift of `0`.

```
function setVault(address _vault) external override onlyLendingPool {
    require(address(vault) == address(0), "84");
    require(IERC4626(_vault).asset() == _underlyingAsset, "83");
    vault = IERC4626(_vault);
    IERC20(_underlyingAsset).forceApprove(address(vault), type(uint256).max);
}
```

Recommendation

The `AToken` contract must allow the admin to modify the `vault` address **when no liquidity is deposited in the current `vault`**.

Resolution

Cod3x team: Fixed in PR14 commits 6b4ddbd and dacc81a.

Zigtur: Fixed. **Warning:** The fix introduces a denial of service vector. Project is aware and is likely to fix this issue.

LOW-01 - Incorrect `MAX_VALID_VOLATILITY_TIER` check in `setVolatilityTier`

Description

Scope:

- ReserveBorrowConfiguration.sol#L155
- ReserveBorrowConfiguration.sol#L39
- MiniPoolBorrowLogic.sol#L123-L129

The `setVolatilityTier` function ensures that the configured value is lower than or equal to 4.

However, the `calculateUserAccountDataVolatile` expect it to be lower than or equal to 2.

Recommendation

Consider setting `MAX_VALID_VOLATILITY_TIER = 2;`.

Resolution

Cod3x team: Fixed in PR14 commit 1cc00ab.

Zigtur: Fixed. The functionality has been removed.

LOW-02 - LTV and liquidation thresholds can be disordered

Description

Scope:

- ReserveBorrowConfiguration.sol#L41-L143

A new feature allows multiple LTV and liquidation thresholds levels. These levels should be in a specific order: - High volatility: The thresholds should be lower than Low and Medium volatility ones. - Medium volatility: The thresholds should be between Low and High volatility ones. - Low volatility: The thresholds should be greater than Medium and High volatility ones.

However, there is no check to ensure that this order is respected.

Recommendation

Consider implementing checks for each function to ensure correct threshold levels.

Resolution

Cod3x team: Fixed in PR14 commit 1cc00ab.

Zigtur: Fixed. The functionality has been removed.

LOW-03 - Discrepancies in LTV and Liquidation Threshold representation

Description

- UserRecentBorrow.sol#L25
- ReserveConfiguration.sol#L49-L51
- ReserveBorrowConfiguration.sol#L37
- MiniPoolBorrowLogic.sol#L123-L129
- MiniPoolBorrowLogic.sol#L179
- MiniPoolValidationLogic.sol#L189-L190
- PercentageMath.sol#L15

The LTV and liquidiation threshold can be set up to `65536`, while it is interpreted as a percentage and should be lower than or equal to `1e4`.

Recommendation

Ensure that LTV and liquidation threshold are not set to a value greater than `1e4`.

Resolution

Cod3x team: Fixed in PR14 commit 1cc00ab.

Zigtur: Fixed. The functionality has been removed.

LOW-04 - Liquidation threshold must be higher than Low, Medium and High volatility liquidation threshold

Description

- ReserveConfiguration.sol#L94-L102

The new feature for multiple LTV and liquidation thresholds levels during borrowing does not apply to liquidations. The liquidation threshold used for liquidations must be higher than the liquidation thresholds set for the low, medium and high volatility levels.

Recommendation

`setLiquidiationThreshold` should ensure that the threshold set is higher than or equal to the highest borrow liquidation threshold level (most likely low volatility one).

Resolution

Cod3x team: Fixed in PR14 commit 1cc00ab.

Zigtur: Fixed. The functionality has been removed.

LOW-05 - AToken reinitialization may overwrite the `_aTokenWrapper` variable

Description

- AToken.sol#L127

The `AToken` contract can be reinitialized. When reinitialized with the same `initialize` function, the `_aTokenWrapper` address will be overwritten.

Recommendation

Consider setting `_aTokenWrapper` only if it has not been already initialized.

```
if (_aTokenWrapper == address(0)) _aTokenWrapper = address(new
↪  ATokenNonRebasing(address(this)));
```

Resolution

Cod3x team: Fixed in PR14 commit 64376e5.

Zigtur: Fixed.

INFO-01 - `Oracle.getAssetPrice` makes external call when not needed

Description

Scope:

- Oracle.sol#L94

The `Oracle.getAssetPrice` always tries to call the `UNDERLYING_ASSET_ADDRESS()` function of the asset parameter. This is used for the mini-pools to retrieve the underlying token of an aToken.

This means that an important assumption is made on all tokens that are not aToken. Non-aToken tokens contract must not implement an `UNDERLYING_ASSET_ADDRESS()` function.

Recommendation

An internal registry handled by the admin can be implemented in the `Oracle` contract. This registry will be used to map the tokens to a boolean indicating if the token is an aToken or not.

Then, `UNDERLYING_ASSET_ADDRESS()` could only be called on aToken contracts.

Resolution

Cod3x team: Acknowledged.

Zigtur: Acknowledged.

INFO-02 - `reserveType` is not documented in Natspec comments

Description

Scope:

- ILendingPool.sol

NatSpec comments for `deposit` and `borrow` do not document `reserveType`.

Recommendation

Document `reserveType` for all functions that use it.

Resolution

Cod3x team: Fixed in PR14 commit 96a3560.

Zigtur: Fixed.

INFO-03 - View function `getTotalManagedAssets` uses an access control check

Description

Scope:

- LendingPool.sol#L725-L733

`LendingPool.getTotalManagedAssets` view function calls the `onlyLendingPoolConfigurator` modifier. Only the pool configurator will be able to call this view function.

Recommendation

Consider removing the access control modifier set on view functions.

Resolution

Cod3x team: Fixed in PR14 commit 96a3560.

Zigtur: Fixed. `onlyLendingPoolConfigurator` modifier has been removed.

INFO-04 - Incorrect interface used in `MiniPool`

Description

Scope:

- MiniPool.sol#L190
- MiniPool.sol#L195
- MiniPool.sol#L342
- MiniPool.sol#L344

`MiniPool` calls the `asset` address multiple times. The interface used is `IAToken` while it should be `ATokenNonRebasing` to avoid confusion.

Recommendation

Replace `IAToken` with `ATokenNonRebasing` to avoid confusion.

Resolution

Cod3x team: Fixed in PR14 commit 96a3560.

Zigtur: Fixed. `ATokenNonRebasing` interface is now used.

INFO-05 - Commented out code

Description

Scope:

- ATokenERC6909.sol#L256-L259

Some code is commented. This code and its impact have not been audited.

Recommendation

Delete the commented out code.

Resolution

Cod3x team: Fixed in PR14 commit 96a3560.

Zigtur: Fixed. Commented out code has been removed.

INFO-06 - Cached variable is not used

Description

Scope:

- MiniPoolDepositLogic.sol#L86
- MiniPoolDepositLogic.sol#L93

Some variable are cached from storage to the stack. However, the cached variable are used and the storage variables are used.

Recommendation

Use the stack variable for gas optimization.

Resolution

Cod3x team: Acknowledged.

Zigtur: Not fixed, acknowledged.

INFO-07 - Claiming rewards is not flexible

Description

Scope:

- RewardsController6909.sol#L56-L57
- RewardsController.sol#L128-L129

Rewards claiming allows chosing the amount of assets to claim.

However, this amount is applied on every `assets`. This is not convenient when a user wants to take different amount for each asset.

Recommendation

Consider using an array of amount, with each amount corresponding to an unique asset.

Resolution

Cod3x team: Acknowledged.

Zigtur: Not fixed, acknowledged.

Appendix

HIGH-01 - Fix patch

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

```
diff --git a/contracts/protocol/core/Oracle.sol
↪  b/contracts/protocol/core/Oracle.sol
index 306e769..f16d1dd 100644
--- a/contracts/protocol/core/Oracle.sol
+++ b/contracts/protocol/core/Oracle.sol
@@ -117,7 +117,7 @@ contract Oracle is IPriceOracleGetter, Ownable {

         // if `asset` is an aToken then convert the price from asset to share.
         if (asset != underlying) {
-            return ATokenNonRebasing(asset).convertToShares(finalPrice);
+            return ATokenNonRebasing(asset).convertToAssets(finalPrice);
         } else {
             return finalPrice;
         }
```

HIGH-02 - Fix patch

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

```
diff --git a/contracts/protocol/core/minipool/MiniPool.sol
↪  b/contracts/protocol/core/minipool/MiniPool.sol
index 21b479f..94b5a32 100644
--- a/contracts/protocol/core/minipool/MiniPool.sol
+++ b/contracts/protocol/core/minipool/MiniPool.sol
@@ -639,7 +639,7 @@ contract MiniPool is VersionedInitializable, IMiniPool,
↪  MiniPoolStorage {
     ) external override onlyMiniPoolConfigurator {
         require(Address.isContract(asset), Errors.LP_NOT_CONTRACT);
         _reserves[asset].init(
-            aTokenAddress, aTokenID, variableDebtTokenID,
↪  interestRateStrategyAddress
+            asset, aTokenAddress, aTokenID, variableDebtTokenID,
↪  interestRateStrategyAddress
         );
         _addReserveToList(asset);
     }
diff --git a/contracts/protocol/core/minipool/logic/MiniPoolReserveLogic.sol
↪  b/contracts/protocol/core/minipool/logic/MiniPoolReserveLogic.sol
index 17067d1..2d75ff3 100644
--- a/contracts/protocol/core/minipool/logic/MiniPoolReserveLogic.sol
+++ b/contracts/protocol/core/minipool/logic/MiniPoolReserveLogic.sol
@@ -169,13 +169,14 @@ library MiniPoolReserveLogic {
      */
     function init(
         DataTypes.MiniPoolReserveData storage reserve,
+        address asset,
         IAERC6909 aTokenAddress,
         uint256 aTokenID,
         uint256 variableDebtTokenID,
         address interestRateStrategyAddress
     ) internal {
         require(
-            aTokenAddress.getUnderlyingAsset(reserve.aTokenID) == address(0),
+            aTokenAddress.getUnderlyingAsset(aTokenID) == asset,
             Errors.RL_RESERVE_ALREADY_INITIALIZED
         );
```

HIGH-04 - Fix patch

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

```
diff --git a/contracts/protocol/core/minipool/logic/MiniPoolLiquidationLogic.sol
↪  b/contracts/protocol/core/minipool/logic/MiniPoolLiquidationLogic.sol
index 0784ee3..caae746 100644
--- a/contracts/protocol/core/minipool/logic/MiniPoolLiquidationLogic.sol
+++ b/contracts/protocol/core/minipool/logic/MiniPoolLiquidationLogic.sol
@@ -27,7 +27,9 @@ library MiniPoolLiquidationLogic {
             abi.encodeWithSignature(
                 "liquidation-
                 ↪  Call(address,bool,address,bool,address,uint256,bool)",
                 params.collateralAsset,
+                false,
                 params.debtAsset,
+                false,
                 params.user,
                 params.debtToCover,
                 params.receiveAToken
```

MEDIUM-03 - Fix patch

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

```
diff --git a/contracts/protocol/libraries/helpers/Errors.sol
↪  b/contracts/protocol/libraries/helpers/Errors.sol
index 2d652aa..abccd36 100644
--- a/contracts/protocol/libraries/helpers/Errors.sol
+++ b/contracts/protocol/libraries/helpers/Errors.sol
@@ -110,6 +110,7 @@ library Errors {
     string public constant VL_FLASHLOAN_DISABLED = "93";
     string public constant LPC_FLASHLOAN_PREMIUM_INVALID = "94";
     string public constant VL_TRANCHED_ASSET_CANNOT_BE_FLASHLOAN = "95";
+    string public constant VL_INVALID_TOKEN = "96";

     enum CollateralManagerErrors {
         NO_ERROR,
diff --git a/contracts/protocol/tokenization/ERC6909/ATokenERC6909.sol
↪  b/contracts/protocol/tokenization/ERC6909/ATokenERC6909.sol
index b23a219..2ef8411 100644
--- a/contracts/protocol/tokenization/ERC6909/ATokenERC6909.sol
+++ b/contracts/protocol/tokenization/ERC6909/ATokenERC6909.sol
@@ -111,6 +111,7 @@ contract ATokenERC6909 is IncentivizedERC6909,
↪  VersionedInitializable {
                 Errors.LP_CALLER_NOT_LENDING_POOL_CONFIGURATOR
         );
         (aTokenID, debtTokenID, isTrancheRet) =
         ↪  getIdForUnderlying(underlyingAsset);
+        require(_underlyingAssetAddresses[aTokenID] == address(0),
↪  Errors.VL_INVALID_TOKEN);
         if (isTrancheRet) {
             _totalTrancheTokens++;
             _isTranche[aTokenID] = true;
```