Competitive Security Assessment

Mobox_LBP

Mar 21th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

• Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.

• Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.

• Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.

• Verify the code base is compliant with the most up-to-date industry standards and security best practices.

• Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

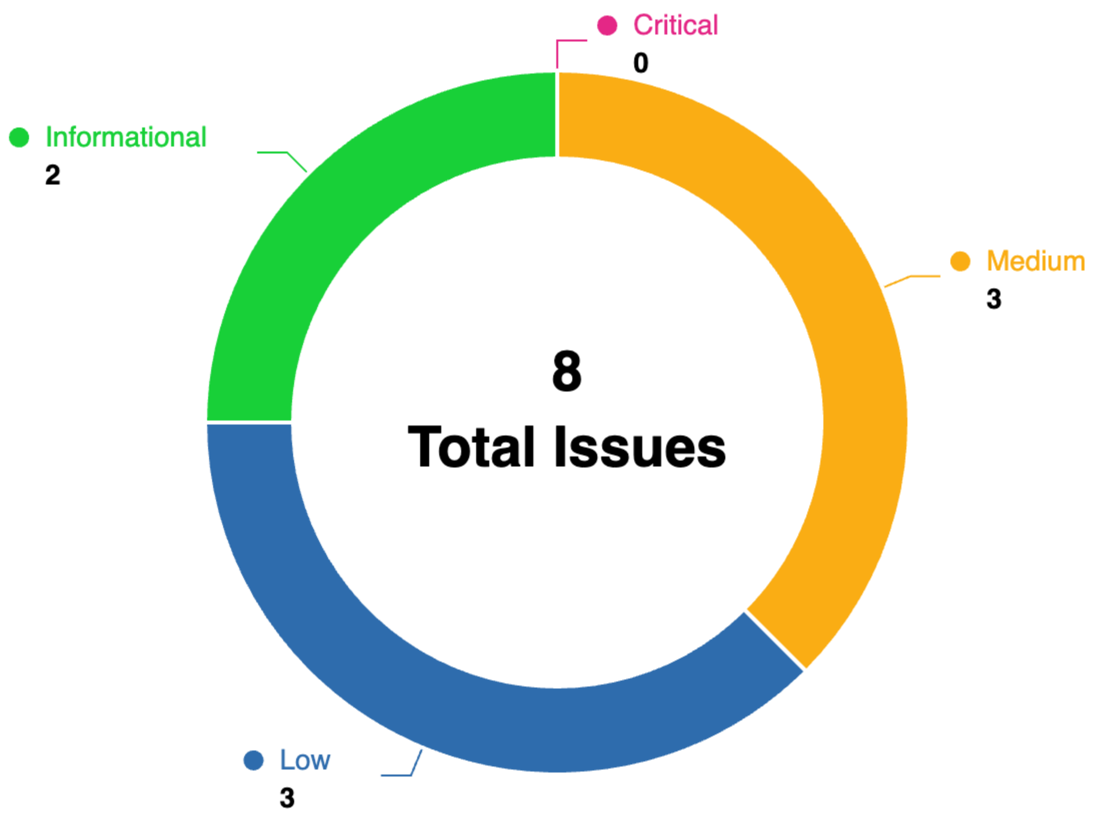
Overview

Project Name	Mobox_LBP
Language	Solidity
Codebase	• https://github.com/moboxlab/ModragonBall_LBP • audit version - 71df06f898a7e83c26611c308a44a111598357a1 • final version - 5c2ab3c8eee27e9b459556dff039d6f136d9a60d
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
./contracts/src/LiquidityBootstrapPool.sol	205d4c9acff362d18c3f3e729e31b967b16b216cec9a66c3868bde829940c045

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
LBP-1	The DOS attack	DOS	Medium	Fixed	danielt
LBP-2	Potential Issue with Zero-Value Transfers in Redeem Function	Logical	Medium	Fixed	8olidity
LBP-3	Fees withdraw in the `close()` function	Logical	Medium	Fixed	danielt
LBP-4	emptyProof can not pass the proof.verify	Logical	Low	Fixed	comcat
LBP-5	Potential Risk of Total Fee Equaling 100% in _swapAssetsForShares Function	Logical	Low	Fixed	8olidity
LBP-6	Centrazliation Risk	Privilege Related	Low	Acknowledged	Hupixiong3, danielt
LBP-7	Unused state variable	Gas Optimization	Informational	Fixed	danielt

LBP-8	Lack of better input validation in `LiquidityBootstrapPool::swapAssetsForExactShares` function	Language Specific	Informational	Fixed	Hupixiong3

LBP-1:The DOS attack

Category	Severity	Client Response	Contributor
DOS	Medium	Fixed	danielt

Code Reference

- code/contracts/src/LiquidityBootstrapPool.sol#L460-L494
- code/contracts/src/LiquidityBootstrapPool.sol#L635-L658

```
460: function _swapAssetsForShares(
461:         address recipient,
462:         address referrer,
463:         uint256 assetsIn,
464:         uint256 sharesOut,
465:         uint256 assets,
466:         uint256 shares,
467:         uint256 swapFees
468:     ) internal virtual recipientIsSender(recipient) {
469:         if (assets + assetsIn - swapFees >= maxTotalAssetsIn()) {
470:             revert AssetsInExceeded();
471:         }
472:
473:         asset().safeTransferFrom(msg.sender, address(this), assetsIn);
474:
475:         uint256 totalPurchasedAfter = totalPurchased + sharesOut;
476:
477:         if (totalPurchasedAfter >= maxTotalSharesOut() || totalPurchasedAfter >= shares) {
478:             revert SharesOutExceeded();
479:         }
480:
481:         totalPurchased = totalPurchasedAfter;
482:
483:         purchasedShares[recipient] = purchasedShares[recipient].rawAdd(sharesOut);
484:
485:         if (referrer != address(0) && referrerFee() != 0) {
486:             uint256 assetsReferred = assetsIn.mulWad(referrerFee());
487:
488:             totalReferred += assetsReferred;
489:
490:             referredAssets[referrer] = referredAssets[referrer].rawAdd(assetsReferred);
491:         }
492:
493:         emit Buy(msg.sender, assetsIn, sharesOut, swapFees);
494:     }
```

```
function close() external virtual {
        if (closed) revert ClosingDisallowed();
        if (block.timestamp < saleEnd()) revert ClosingDisallowed();

        uint256 totalAssets = asset().balanceOf(address(this)).rawSub(totalSwapFeesAsset);
        uint256 platformFees = totalAssets.mulWad(platformFee());
        uint256 totalAssetsMinusFees = totalAssets.rawSub(platformFees).rawSub(totalReferred);

        if (totalAssets != 0) {
            // Transfer asset
            asset().safeTransfer(manager(), totalAssetsMinusFees);
        }

        uint256 totalShares = share().balanceOf(address(this));
        uint256 unsoldShares = totalShares.rawSub(totalPurchased);

        if (totalShares != 0) {
            share().safeTransfer(manager(), unsoldShares);
        }

        closed = true;

        emit Close(totalAssetsMinusFees, platformFees, totalSwapFeesAsset, totalSwapFeesShare);
    }
```

Description

danielt: When swapping assets for shares, the **`_swapAssetsForShares`** function will be executed, and the referrer will get a referrer fee.
The referrer can withdraw his/her referrer fee when redeeming with the **`redeem`** function.
However, in the **`_swapAssetsForShares`** function, the referrer fee is kind of minted instead of deducted from the input asset:

```
        if (referrer != address(0) && referrerFee() != 0) {
            uint256 assetsReferred = assetsIn.mulWad(referrerFee());

            totalReferred += assetsReferred;

            referredAssets[referrer] = referredAssets[referrer].rawAdd(assetsReferred);
        }
```

As a result, a malicious user can repeatedly swap assets for shares and swap inversely, to increase his/her referrer fee without limit, and the **`totalReferred`** will also become an extremely big value, that is greater than the **`totalAssets`**, which results in closing the pool fail, because the below calculation with the **`rawSub`** will underflow and the asset transfer will fail and revert:

```
    function close() external virtual {
        ...
        uint256 totalAssetsMinusFees = totalAssets.rawSub(platformFees).rawSub(totalReferred);

        if (totalAssets != 0) {
            // Transfer asset
            asset().safeTransfer(manager(), totalAssetsMinusFees);
        }
        ...
}
```

Recommendation

danielt: Recommend deducting the referrer fee from the input asset or limiting the referrer fee to a valid range.

Client Response

danielt: Fixed,New:Add overflow check to prevent overflow from affecting transactions
https://github.com/moboxlab/ModragonBall_LBP/commit/3ea8d2990fb1a5698b2b00948e90344101e86965
Old:Delete content related to handling fees(Has been restored:
https://github.com/moboxlab/ModragonBall_LBP/commit/3ea8d2990fb1a5698b2b00948e90344101e86965)
https://github.com/moboxlab/ModragonBall_LBP/commit/0df4814f315bb1b8ba9b78ab22c8a01e2b0b0f67

LBP-2:Potential Issue with Zero-Value Transfers in Redeem Function

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	8olidity

Code Reference

- code/contracts/src/LiquidityBootstrapPool.sol#L673-L696

```
673: function redeem(
674:         address recipient,
675:         bool referred
676:     ) external virtual recipientIsSender(recipient) returns (uint256 shares) {
677:         if (!closed) revert RedeemingDisallowed();
678:
679:         shares = purchasedShares[msg.sender];
680:
681:         delete purchasedShares[msg.sender];
682:
683:         share().safeTransfer(msg.sender, shares);
684:
685:         if (referred && referrerFee() != 0) {
686:             uint256 assets = referredAssets[msg.sender];
687:
688:             delete referredAssets[msg.sender];
689:
690:             asset().safeTransfer(recipient, assets);
691:         }
692:
693:         if (shares != 0) {
694:             emit Redeem(msg.sender, block.timestamp, shares);
695:         }
696:     }
```

Description

8olidity: The smart contract function redeem is designed to handle the redemption of assets by users. However, there is a potential issue when a user attempts to redeem with zero shares. The ERC20 token standard implemented by share() may not allow zero-value transfers, which could cause the redeem function to fail when safeTransfer is called with a zero balance. This would prevent users without shares from being able to redeem their entitled assets.

```
    function redeem(
        address recipient,
        bool referred
    ) external virtual recipientIsSender(recipient) returns (uint256 shares) {
        if (!closed) revert RedeemingDisallowed();

        shares = purchasedShares[msg.sender];

        delete purchasedShares[msg.sender];

        share().safeTransfer(msg.sender, shares);//@audit

        if (referred && referrerFee() != 0) {
            uint256 assets = referredAssets[msg.sender];

            delete referredAssets[msg.sender];

            asset().safeTransfer(recipient, assets);
        }

        if (shares != 0) {
            emit Redeem(msg.sender, block.timestamp, shares);
        }
    }
```

Recommendation

8olidity: To ensure that the contract functions correctly even when users have zero shares, the redeem function should be modified to check the balance before attempting a transfer. The safeTransfer call should only be made if the number of shares is greater than zero. This avoids the attempt of a zero-value transfer and ensures that users can still redeem any other assets they are entitled to without interruption.

Client Response

8olidity: Fixed,Check whether the amount the user can receive is 0
https://github.com/moboxlab/ModragonBall_LBP/commit/714060e88baea5c7630fb0c34edfa475ef800025

LBP-3:Fees withdraw in the `close()` function

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	danielt

Code Reference

- code/contracts/src/LiquidityBootstrapPool.sol#L639-L641

```
639: uint256 totalAssets = asset().balanceOf(address(this)).rawSub(totalSwapFeesAsset);
640:         uint256 platformFees = totalAssets.mulWad(platformFee());
641:         uint256 totalAssetsMinusFees = totalAssets.rawSub(platformFees).rawSub(totalReferred);
```

Description

danielt: The `` `close` `` function closes the pool, and charges swap fees, and platform fees. However, there is no recipient to receive those fees, which results in those fees being locked in the contract.
There is an emergency function that can transfer all the assets and shares of the contract, however, it is only for emergency cases, and invoking the emergency function has a side effect on users' redemption too if there is any user has not redeemed. Thus, it is necessary to send the platform fee to a recipient or use an extra function to claim the platform fee.

Recommendation

danielt: Consider sending those fees to the recipient or adding a function to claim the platform fee.

Client Response

danielt: Fixed,New: Added a method to collect handling fees, which can only be used after closing
https://github.com/moboxlab/ModragonBall_LBP/commit/5c2ab3c8eee27e9b459556dff039d6f136d9a60d
Old: Add all fees to manager balance (Has been
restored:https://github.com/moboxlab/ModragonBall_LBP/commit/5126c163ca6845a9c814e28d467dea0c94e83fa4)
https://github.com/moboxlab/ModragonBall_LBP/commit/7f8bce01788eacb50453564fd25c393f6101345c

LBP-4:emptyProof can not pass the proof.verify

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	comcat

Code Reference

- code/contracts/src/LiquidityBootstrapPool.sol#L399

```
399: return swapAssetsForExactShares(sharesOut, maxAssetsIn, recipient, referrer, MerkleProofLib.emp
tyProof());
```

Description

comcat:

```
function swapAssetsForExactShares(
        uint256 sharesOut,
        uint256 maxAssetsIn,
        address recipient,
        address referrer
    ) external virtual returns (uint256 assetsIn) {
        return swapAssetsForExactShares(sharesOut, maxAssetsIn, recipient, referrer, MerkleProofLi
b.emptyProof());
    }
```

it will pass to the modifier `onlyWhitelisted(proof)`
and then it will call:

```
if (!proof.verify(whitelistMerkleRoot(), keccak256(abi.encodePacked(msg.sender)))) {
                revert WhitelistProof();
            }
```

however, inside the verify function, if the length of proof is zero, then it will return `root == keccak(msg.sender)`. if the root is set to non-zero, it will revert.

```
function verify(bytes32[] memory proof, bytes32 root, bytes32 leaf)
        internal
        pure
        returns (bool isValid)
    {
        /// @solidity memory-safe-assembly
        assembly {
            if mload(proof) {
                // Initialize `offset` to the offset of `proof` elements in memory.
                let offset := add(proof, 0x20)
                // Left shift by 5 is equivalent to multiplying by 0x20.
                let end := add(offset, shl(5, mload(proof)))
                // Iterate over proof elements to compute root hash.
                for {} 1 {} {
                    // Slot of `leaf` in scratch space.
                    // If the condition is true: 0x20, otherwise: 0x00.
                    let scratch := shl(5, gt(leaf, mload(offset)))
                    // Store elements to hash contiguously in scratch space.
                    // Scratch space is 64 bytes (0x00 - 0x3f) and both elements are 32 bytes.
                    mstore(scratch, leaf)
                    mstore(xor(scratch, 0x20), mload(offset))
                    // Reuse `leaf` to store the hash to reduce stack operations.
                    leaf := keccak256(0x00, 0x40)
                    offset := add(offset, 0x20)
                    if iszero(lt(offset, end)) { break }
                }
            }
            isValid := eq(leaf, root)
        }
    }
```

Recommendation

comcat: if it is empty proof, it should return true by default. and it should pass the **`onlyWhitelisted`**

```
modifier onlyWhitelisted(bytes32[] memory proof) virtual {
        if (whitelisted()) {
            if (proof.length != 0 && !proof.verify(whitelistMerkleRoot(), keccak256(abi.encodePack
ed(msg.sender)))) {
                revert WhitelistProof();
            }
        }
        _;
    }
```

Client Response

comcat: Fixed,Add poorf length check
https://github.com/moboxlab/ModragonBall_LBP/commit/7621f17cfbad5e324ff597dccaca2eb53083cc99

LBP-5:Potential Risk of Total Fee Equaling 100% in _swapAssetsForShares Function

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	8olidity

Code Reference

- code/contracts/src/LiquidityBootstrapPool.sol#L421-L428

```
421: uint256 swapFees = assetsIn.mulWad(swapFee());
422:         totalSwapFeesAsset += swapFees;
423:
424:         sharesOut = pool.previewSharesOut(assetsIn.rawSub(swapFees));
425:
426:         if (sharesOut < minSharesOut) revert SlippageExceeded();
427:
428:         _swapAssetsForShares(recipient, referrer, assetsIn, sharesOut, pool.assets, pool.shares,
swapFees);
```

Description

8olidity: In the `_swapAssetsForShares` function, there is a potential risk where the sum of `swapFee()` and `referrerFee()` could equal to 1e18 (100%). In this case, all of the `assetsIn` would be used to pay for the fees, and the user would not receive any tokens.
The `_swapAssetsForShares` function calculates the swap fees and referral fees using the following lines of code:

```
uint256 swapFees = assetsIn.mulWad(swapFee());
uint256 assetsReferred = assetsIn.mulWad(referrerFee());
```

This means that all of the assets would be used to pay for the fees, and the user would not receive any tokens.
The `mulWad` function is used to perform fixed-point multiplication, where the fees (represented as fractions with a denominator of 1e18) are multiplied with the `assetsIn` amount.
In the extreme case where `swapFee() + referrerFee() = 1e18`, the following would occur:

```
swapFees + assetsReferred
= assetsIn.mulWad(swapFee()) + assetsIn.mulWad(referrerFee())
= assetsIn.mulWad(swapFee() + referrerFee())
= assetsIn.mulWad(1e18)
= assetsIn
```

Recommendation

8olidity: To mitigate this risk, it is recommended to add a check in the _swapAssetsForShares function to ensure that the sum of swapFee() and referrerFee() does not exceed 1e18. For example:

```
require(swapFee() + referrerFee() < 1e18, "Total fee cannot exceed 100%");
```

Client Response

8olidity: Fixed,Add a check to prevent the renewal fee from being greater than or equal to 100%
https://github.com/moboxlab/ModragonBall_LBP/commit/2910bc33b393304168d9e91ce9dc9969d50fedfe

LBP-6:Centrazliation Risk

Category	Severity	Client Response	Contributor
Privilege Related	Low	Acknowledged	Hupixiong3, danielt

Code Reference

- code/contracts/src/LiquidityBootstrapPool.sol#L705-L724
- code/contracts/src/LiquidityBootstrapPool.sol#L716-L724

```
705: function togglePause() external virtual {
706:         if (msg.sender != manager()) {
707:             revert CallerDisallowed();
708:         }
709:
710:         _togglePause();
711:     }
712:
713:     /// @notice Emergency withdrawal and pause function.
714:     /// @dev This method is used for emergency withdrawals from the project side.
715:     /// @param recipient The address to receive redeemed shares and assets.
716:     function emergencyWithdrawal(address recipient) external {
717:         if (msg.sender != manager()) {
718:             revert CallerDisallowed();
719:         }
720:         if (paused == false) _togglePause();
721:
722:         share().safeTransfer(recipient, share().balanceOf(address(this)));
723:         asset().safeTransfer(recipient, asset().balanceOf(address(this)));
724:     }
```

```
716: function emergencyWithdrawal(address recipient) external {
717:         if (msg.sender != manager()) {
718:             revert CallerDisallowed();
719:         }
720:         if (paused == false) _togglePause();
721:
722:         share().safeTransfer(recipient, share().balanceOf(address(this)));
723:         asset().safeTransfer(recipient, asset().balanceOf(address(this)));
724:     }
```

Description

Hupixiong3: The emergencyWithdrawal function qualifies the manager address to be invoked, but the comment does not indicate whether the address is a multi-signed address. If it is just an ordinary EOA address, a private key leak could deplete pool assets.
danielt: The privileged role manager can pause the contract, and transfer all the share tokens and the asset tokens of the contract to an arbitrary address, with the `emergencyWithdrawal` function.
If the private key of the manager is compromised, all the assets of the contract will be at risk.

Recommendation

Hupixiong3: You are advised to use the multi-signature address as the manager address.
danielt: Recommend applying an extra decentralized mechanism like multi-signature, and timelock to mitigate the centralization risk

Client Response

Hupixiong3: Acknowledged
danielt: Acknowledged

LBP-7:Unused state variable

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	danielt

Code Reference

- code/contracts/src/LiquidityBootstrapPool.sol#L108

```
108: mapping(address => uint256) public redeemedShares;
```

Description

danielt: The state variable `redeemedShares` is defined but never used.

Recommendation

danielt: Recommend removing the unused variable.

Client Response

danielt: Fixed,
Useless variables have been deleted
https://github.com/moboxlab/ModragonBall_LBP/commit/a3027cf398d56c335c8e57ff2040813bf1244e11

LBP-8:Lack of better input validation in `LiquidityBootstrapPool::swapAssetsForExactShares` function

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	Hupixiong3

Code Reference

- code/contracts/src/LiquidityBootstrapPool.sol#L441-L458

```
441: function swapAssetsForExactShares(
442:         uint256 sharesOut,
443:         uint256 maxAssetsIn,
444:         address recipient,
445:         address referrer,
446:         bytes32[] memory proof
447:     ) public virtual whenNotPaused whenSaleActive onlyWhitelisted(proof) nonReentrant returns (uint256 assetsIn) {
448:         Pool memory pool = args();
449:
450:         assetsIn = pool.previewAssetsIn(sharesOut);
451:         uint256 swapFees = assetsIn.mulWad(swapFee());
452:         assetsIn = assetsIn.rawAdd(swapFees);
453:         totalSwapFeesAsset += swapFees;
454:
455:         if (assetsIn > maxAssetsIn) revert SlippageExceeded();
456:
457:         _swapAssetsForShares(recipient, referrer, assetsIn, sharesOut, pool.assets, pool.shares, swapFees);
458:     }
```

Description

Hupixiong3: In addition to the verification of proof, there is no verification of other parameters. Validation of other parameters reduces security risks and the possibility of misoperation.

Recommendation

Hupixiong3: Add validation logic for other parameters.

```
function swapExactAssetsForShares(
    uint256 assetsIn,
    uint256 minSharesOut,
    address recipient,
    address referrer,
    bytes32[] memory proof
) public {
    require(assetsIn > 0, "AssetsIn must be greater than 0");
    require(minSharesOut > 0, "minSharesOut must be greater than 0");
    require(recipient != address(0), "Recipient cannot be the zero address");
    // Other code is omitted...
}
```

Client Response

Hupixiong3: Fixed,swap method increases quantity judgment
https://github.com/moboxlab/ModragonBall_LBP/commit/44197f3feb9c2d12df43f086e5406d917491bddb

Disclaimer