Protocol Audit Report

Prepared by: Bizarro

Table of Contents

Protocol Summary

In the realm of cryptocurrency, liquidity is paramount, especially for nascent Meme projects. Liquidity refers to the ease with which assets can be bought or sold in the market without affecting their price. For cryptocurrency projects, having sufficient liquidity is crucial for several reasons.

To search for an efficient, near-zero cost liquidity solution for token launch, we propose a straightforward mechanism: the concept of virtual liquidity to meet the liquidity needs of project parties. We will establish a deep liquidity pool on UniswapV3 to satisfy the liquidity exit for users' buying and selling activities. Essentially, this mechanism involves whales providing liquidity to retail investors. Liquidity providers can earn profits through LP fees, creating a win-win situation where whales earn LP fees and retail investors/developers resolve their liquidity issues.

We believe this mechanism can enhance the liquidity returns for DeFi whales while simultaneously addressing the liquidity challenges faced by retail investors and developers.

Disclaimer

Bizarro found as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Scope

See scope.txt

Roles

Executive Summary

Issues found

Severity	**Number of issues found**
High	0
Medium	0
Low	6
Info	3
Gas	0
Total	9

Findings

Low

[L-1] The `LamboFactory::createLaunchPad` should contain a check when minting pool tokens.

Description: In `LamboFactory::createLaunchPad` function when minting the pool token from the `IPool` contract, the minting process does not confirm whether liquidity was successfully minted. Specifically, the return value of the `IPool(pool).mint(address(this))` call is not checked.

```
    IPool(pool).mint(address(this));
    IERC20(pool).safeTransfer(address(0),
IERC20(pool).balanceOf(address(this)));
```

Impact: The lack of validation for successful liquidity minting introduces

1. Liquidity Mismanagement.
2. Inaccurate State Reporting.
3. Token burning without Minting.

Recommended Mitigation:

```
-           IPool(pool).mint(address(this));
+           uint256 liq = IPool(pool).mint(address(this));
+           require(liq > 0, "Minting token failed");
```

[L-2] Unnecessary Use of `Ownable` in `LamboToken` contract.

Description: The `LamboToken` contract unnecessarily inherits the `Ownable` contract from OpenZeppelin. This introduces extra complexity and gas costs without adding any value, as the ownership of the contract is transferred to 0 address.

Impact:

1. Increase Gas Costs: The inclusion of `Ownable` unnecessarily increases the bytecode size and execution costs.
2. Wasted Code Complexity: The contract contains unused functionality, making it harder to maintain and understand.
3. Risk of Incorrect Assumptions: Developers or auditors might mistakenly assume that ownership mechanisms are active and can be used for administrative purposes.

Recommended Mitigation:

```
    contract LamboToken is Context, IERC20, IERC20Metadata, IERC20Errors,
-     Ownable
     {
        mapping(address account => uint256) private _balances;

        mapping(address account => mapping(address spender => uint256))
private _allowances;

        uint256 private _totalSupply;
        string public __name;
        string public __symbol;

        /**
        * @dev Sets the values for {name} and {symbol}.
        *
        * All two of these values are immutable: they can only be set once
during
        * construction.
        */
        constructor()
-        Ownable(msg.sender)
        {
-           _transferOwnership(address(0));
        }

        function initialize(string memory _name, string memory _symbol)
public {
            require(_totalSupply == 0, "LamboToken: Already initialized");

            __name = _name;
            __symbol = _symbol;

            _mint(msg.sender, LaunchPadUtils.TOTAL_AMOUNT_OF_QUOTE_TOKEN);
-           _transferOwnership(address(0));
        }
```

[L-3] Event Parameters are wrong in the `LamboVEthRouter` contract.

Description: The `BuyQuote` event in the `LamboVEthRouter` contract is emitting an incorrect value for the amountXOut parameter, which should instead be amountYOut.

Impact: This discrepancy can lead to confusion for off-chain systems or users relying on event data for computations or auditing purposes.

Proof of Concept:

```


@>      event BuyQuote(address quoteToken, uint256 amountXIn, uint256
amountXOut);

            function _buyQuote(address quoteToken, uint256 amountXIn,
uint256 minReturn) internal returns (uint256 amountYOut) {

            require(msg.value >= amountXIn, "Insufficient msg.value");

            // handle fee
            uint256 fee = (amountXIn * feeRate) / feeDenominator;
            amountXIn = amountXIn - fee;
            (bool success, ) = payable(owner()).call{value: fee}("");
            require(success, "Transfer to Owner failed");
            // handle swap
            address pair =
UniswapV2Library.pairFor(LaunchPadUtils.UNISWAP_POOL_FACTORY_, vETH,
quoteToken);
            (uint256 reserveIn, uint256 reserveOut) =
UniswapV2Library.getReserves(
                LaunchPadUtils.UNISWAP_POOL_FACTORY_,
                vETH,
                quoteToken
            );

            // Calculate the amount of quoteToken to be received
            amountYOut = UniswapV2Library.getAmountOut(amountXIn,
reserveIn, reserveOut);
            require(amountYOut >= minReturn, "Insufficient output
amount");

            // Transfer vETH to the pair
            VirtualToken(vETH).cashIn{value: amountXIn}(amountXIn);
            IERC20(vETH).safeTransfer(pair, amountXIn);

            // Perform the swap
            (uint256 amount0Out, uint256 amount1Out) = vETH < quoteToken
                ? (uint256(0), amountYOut)
                : (amountYOut, uint256(0));

            IUniswapV2Pair(pair).swap(amount0Out, amount1Out, msg.sender,
new bytes(0));


            if (msg.value > (amountXIn + fee + 1)) {
                (bool success, ) = payable(msg.sender).call{value:
msg.value - amountXIn - fee - 1}("");

                require(success, "ETH transfer failed");
            }
@>              emit BuyQuote(quoteToken, amountXIn, amountYOut);
        }
```

Recommended Mitigation:

```diff
-        event BuyQuote(address quoteToken, uint256 amountXIn, uint256
amountXOut);
+        event BuyQuote(address quoteToken, uint256 amountXIn, uint256
amountYOut);
```

[L-4] `receive()` Function Being Payable.

Description: In `LamboVEthRouter` & `LamboRebalanceOnUniwap` has payable `receive()` function, The purpose of a receive() function is typically to allow the contract to accept Ether sent directly to it without any accompanying function call. However, in this contract, the intended behavior does not clearly necessitate the accumulation of Ether in the contract itself.

Impact: Ether sent directly to the contract may become locked, creating a risk of funds being unintentionally trapped.

Proof of Concept:

```
@>    receive() external payable {}
```

Recommended Mitigation:

```
1.

-    receive() external payable {}
+    receive() external {}
```

```
2.

    receive() external payable {}

+    function withdrawETH() external onlyOwner {
+    uint256 balance = address(this).balance;
+    require(balance > 0, "No ETH to withdraw");
+    payable(owner()).transfer(balance);
+    }
```

[L-5] `VirtualToken::takeLoan` Should not be payable as it does not handle eth in the function.

Description: The `takeLoan()` function is marked `payable`, allowing Ether to be sent along with the function call. The function does not use or process msg.value. There is no logic to handle, refund, or account for Ether sent with the call. Marking a function as payable without any Ether-related logic can lead to confusion for developers and users interacting with the contract. Ether sent to the function would remain trapped in the contract unless additional withdrawal functionality is implemented.

Impact: Confusion and inefficiency in contract usage. Any Ether mistakenly sent to the function would be locked in the contract, potentially leading to user dissatisfaction.

Proof of Concept:

```
    function takeLoan(address to, uint256 amount) external
@>     payable
     onlyValidFactory {
        if (block.number > lastLoanBlock) {
            lastLoanBlock = block.number;
            loanedAmountThisBlock = 0;
        }
        require(loanedAmountThisBlock + amount <= MAX_LOAN_PER_BLOCK,
"Loan limit per block exceeded");

        loanedAmountThisBlock += amount;

        _mint(to, amount);
        _increaseDebt(to, amount);

        emit LoanTaken(to, amount);
    }
```

Recommended Mitigation:

```
    function takeLoan(address to, uint256 amount) external
-     payable
     onlyValidFactory {
        if (block.number > lastLoanBlock) {
            lastLoanBlock = block.number;
            loanedAmountThisBlock = 0;
        }
        require(loanedAmountThisBlock + amount <= MAX_LOAN_PER_BLOCK,
"Loan limit per block exceeded");

        loanedAmountThisBlock += amount;

        _mint(to, amount);
        _increaseDebt(to, amount);

        emit LoanTaken(to, amount);
    }
```

[L-6] In `VirtualToken::_transferAssset` there is a check for comparing the underlying token to the native token of the contract.

Description: The `_transferAssetFromUser` function performs redundant checks for the type of `underlyingToken` and validates the `msg.value` parameter even though these checks are already handled in the calling function, `cashIn`. This introduces unnecessary complexity and slight inefficiency in the smart contract code.

Impact:

1. Redundant code increases gas usage slightly.
2. Code readability and maintainability are affected, increasing the likelihood of bugs or misinterpretation.

Proof of Concept:

```
    function _transferAssetFromUser(uint256 amount) internal {

@>       if (underlyingToken == LaunchPadUtils.NATIVE_TOKEN) {
            require(msg.value >= amount, "Invalid ETH amount");
        } else {
            IERC20(underlyingToken).safeTransferFrom(msg.sender,
address(this), amount);
        }
    }
```

Recommended Mitigation:

```

function _transferAssetFromUser(uint256 amount) internal {

-       if (underlyingToken == LaunchPadUtils.NATIVE_TOKEN) {
-            require(msg.value >= amount, "Invalid ETH amount");
-        } else {
            IERC20(underlyingToken).safeTransferFrom(msg.sender,
address(this), amount);
-       }
    }

```

Info

[I-1] The `LaunchPadUtils::MAX_AMOUNT` should not be hardcoded

Description: The `LaunchPadUtils::MAX_AMOUNT` should not be hardcoded to avoid human mistakes when creating a smart contract.

Impact: Major Changes in MAX_AMOUNT

Proof of Concept:

```
@>    uint256 public constant MAX_AMOUNT =
11579208923731619542357098500868790785326998466564056403945758400791312963
```

```
9935;
```

Recommended Mitigation:

```diff
-       uint256 public constant MAX_AMOUNT =
115792089237316195423570985008687907853269984665640564039457584007913129639935;
+       uint256 public constant MAX_AMOUNT = type(uint256).max
```

[I-2] `VirtualToken::cashIn` function should be nonReentrant.

Description: Although the `cashIn` function follows checks effects and interactions, to avoid reentrancy the function should add `openzeppelin`'s nonReentrant modifier.

Recommended Mitigation:

```diff
    function cashIn(uint256 amount)
+    nonReentrant
     external
     payable
     onlyWhiteListed {
        if (underlyingToken == LaunchPadUtils.NATIVE_TOKEN) {
            require(msg.value == amount, "Invalid ETH amount");
        } else {
            _transferAssetFromUser(amount);
        }
        _mint(msg.sender, msg.value);
        emit CashIn(msg.sender, msg.value);
    }
```

[I-3] Unused import in `LamboFactory`

Description: in `LamboFactory.sol` the `LamboVEthRouter` import is unused.

Recommended Mitigation:

```diff
-   import {LamboVEthRouter} from "./LamboVEthRouter.sol";
```