Rango V2.1

Security Audit Report

September 25, 2024

Contents

1 Introduction

1.1 About Rango

`Rango` is a new layer on top of all Bridges and DEXs, working as a Bridge Aggregator and DEX Aggregator at the same time to enable seamless on-chain and cross-chain swaps, finding the most efficient, safe, cheap and fast route for swapping from any token on any blockchain to any other token to any blockchain.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/rango-exchange/rango-contracts-v2
- CommitID: ec28303

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/rango-exchange/rango-contracts-v2
- CommitID: 579d864

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `Rango V2.1` protocol. Throughout this audit, we identified a total of 5 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	-	-	-	-
Medium	3	-	-	3
Low	2	1	-	1
Informational	-	-	-	-
Undetermined	-	-	-	-

3 | Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

3.3.1 [M-1] Possible Revert in LibSwapper::callSwapsAndFees()

Target	Category	IMPACT	LIKELIHOOD	STATUS
LibSwapper.sol	Business Logic	Medium	Medium	Addressed

The `callSwapsAndFees()` function is designed to swap the user-specified `fromToken` into `toToken` on the source chain. It also handles charging a swap fee through the `collectFeesForSwap()` function (line 285). By design, this fee can be taken in either the `fromToken` or the `toToken`.

While examining its implementation, we observe that the fee is collected (line 285) before executing the token swap (line 296). If the fee is meant to be collected in the `toToken` (line 358), the transaction will be reverted. This is because the contract does not hold any `toToken` before the swap is performed.

LibSwapper::callSwapsAndFees()

```
281 function callSwapsAndFees(SwapRequest memory request, Call[] calldata calls)
        private returns (bytes[] memory) {
282     ...

284     // Get Fees
285     LibSwapper.collectFeesForSwap(request);

287     // Execute swap Calls
288     bytes[] memory returnData = new bytes[](calls.length);
289     address tmpSwapFromToken;
290     for (uint256 i = 0; i < calls.length; i++) {
291         tmpSwapFromToken = calls[i].swapFromToken;
292         bool isTokenNative = tmpSwapFromToken == ETH;
293         if (isTokenNative == false)
294             approveMax(tmpSwapFromToken, calls[i].spender, calls[i].amount);

296         (bool success, bytes memory ret) = isTokenNative
297         ? calls[i].target.call{value : calls[i].amount}(calls[i].callData)
298         : calls[i].target.call(calls[i].callData);

300         emit CallResult(calls[i].target, success, ret);
301         if (!success)
302             revert(_getRevertMsg(ret));
303         returnData[i] = ret;
304     }
305     ...
306 }
```

LibSwapper::collectFeesForSwap()

```
352 function collectFeesForSwap(SwapRequest memory request) internal {
353     BaseSwapperStorage storage baseSwapperStorage = getBaseSwapperStorage();
354     // Get Platform fee
355     bool hasPlatformFee = request.platformFee > 0;
356     bool hasDestExecutorFee = request.destinationExecutorFee > 0;
357     bool hasAffiliateFee = request.affiliateFee > 0;
358     address feeToken = request.feeFromInputToken ? request.fromToken : request.
            toToken;
359     if (hasPlatformFee  hasDestExecutorFee) {
360         require(baseSwapperStorage.feeContractAddress != ETH, "Fee contract
                address not set");
361         _sendToken(feeToken, request.platformFee + request.
                destinationExecutorFee, baseSwapperStorage.feeContractAddress,
                feeToken == ETH, false);
362     }

364     // Get affiliate fee
365     if (hasAffiliateFee) {
366         require(request.affiliatorAddress != ETH, "Invalid affiliatorAddress");
367         _sendToken(feeToken, request.affiliateFee, request.affiliatorAddress,
                feeToken == ETH, false);
368     }

370     ...
371 }
```

Remediation The `callSwapsAndFees()` function should be called after the swap operation has been completed.

3.3.2 [M-2] Improper Swap-out Amount in LibSwapper::onChainSwapsInternal()

Target	Category	IMPACT	LIKELIHOOD	STATUS
LibSwapper.sol	Business Logic	Medium	Medium	Addressed

The `onChainSwapsInternal()` function is responsible for swapping the user-specified `fromToken` into `toToken` and returning the resulting amount of `toToken`. It performs the token swap and charges the swap fee by calling the `callSwapsAndFees()` function (line 247). Especially, the amount of `toToken` received (i.e., `secondaryBalance`) is calculated by checking the contract's `toToken` balance before and after the swap process (line 259).

While examining its implementation, it turns out that if the swap fee is taken in the `toToken`, it incorrectly subtracts the swap fee from the `secondaryBalance` (line 262). This is erroneous because

`secondaryBalance` already accounts for the swap fee deduction, resulting in a double deduction.

LibSwapper::onChainSwapsInternal()

```
233 function onChainSwapsInternal(
234     SwapRequest memory request,
235     Call[] calldata calls,
236     uint256 extraNativeFee
237 ) internal returns (bytes[] memory, uint) {

239     uint toBalanceBefore = getBalanceOf(request.toToken);
240     uint fromBalanceBefore = getBalanceOf(request.fromToken);
241     uint256[] memory initialBalancesList = getInitialBalancesList(calls);

243     // transfer tokens from user for SwapRequest and Calls that require transfer
            from user.
244     transferTokensFromUserForSwapRequest(request);
245     transferTokensFromUserForCalls(calls);

247     bytes[] memory result = callSwapsAndFees(request, calls);

249     // check if any extra tokens were taken from contract and return excess
            tokens if any.
250     returnExcessAmounts(request, calls, initialBalancesList);

252     // get balance after returning excesses.
253     uint fromBalanceAfter = getBalanceOf(request.fromToken);

255     ...

257     uint toBalanceAfter = getBalanceOf(request.toToken);

259     uint secondaryBalance = toBalanceAfter - toBalanceBefore;
260     require(secondaryBalance >= request.minimumAmountExpected, "Output is less
            than minimum expected");

262     return (result, secondaryBalance - (request.feeFromInputToken ? 0 : sumFees(
            request)));
263 }
```

Remediation Adjust the calculation to prevent the double deduction and accurately reflect the correct amount of `toToken`.

3.3.3 [M-3] Revisited Logic of LibSwapper::onChainSwapsPreBridge()

Target	Category	IMPACT	LIKELIHOOD	STATUS
LibSwapper.sol	Business Logic	Medium	Medium	Addressed

The `onChainSwapsPreBridge()` function is designed to perform a token swap on the source chain before bridging. Within this function, the minimum amount of native token required is checked at lines 218-219. However, the calculation of `minimumRequiredValue` does not account for the scenario where `toToken` is used as the fee token. This oversight can lead to the minimum check triggering a revert in such cases.

LibSwapper::onChainSwapsPreBridge()

```
211 function onChainSwapsPreBridge(
212     SwapRequest memory request,
213     Call[] calldata calls,
214     uint extraFee
215 ) internal returns (uint out) {

217     bool isNative = request.fromToken == ETH;
218     uint minimumRequiredValue = (isNative ? request.platformFee + request.
            affiliateFee + request.amountIn + request.destinationExecutorFee : 0) +
            extraFee;
219     require(msg.value >= minimumRequiredValue, 'Send more ETH to cover input
            amount + fee');

221     (, out) = onChainSwapsInternal(request, calls, extraFee);
222     // when there is a bridge after swap, set the receiver in swap event to
            address(0)
223     emitSwapEvent(request, out, ETH);

225     return out;
226 }
```

Remediation When `request.feeFromInputToken==false`, the swap fee should not be added into `minimumRequiredValue`.

3.3.4 [L-1] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Low	Low	Acknowledged

In the `Rango V2.1` protocol, the existence of a privileged `owner` account introduces centralization risks, as it holds significant control and authority over critical operations governing the protocol. In

the following, we show the representative function potentially affected by the privileges associated with the privileged account.

DiamondCutFacet::diamondCut()

```
contract DiamondCutFacet is IDiamondCut {
    /// @notice Add/replace/remove any number of functions and optionally execute
    ///         a function with delegatecall
    /// @param _diamondCut Contains the facet addresses and function selectors
    /// @param _init The address of the contract or facet to execute _calldata
    /// @param _calldata A function call, including function selector and
        arguments
    ///                  _calldata is executed with delegatecall on _init
    function diamondCut(
        FacetCut[] calldata _diamondCut,
        address _init,
        bytes calldata _calldata
    ) external override {
        LibDiamond.enforceIsContractOwner();
        LibDiamond.diamondCut(_diamondCut, _init, _calldata);
    }
}
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team This issue has been confirmed by the team.

3.3.5 [L-2] Improper Event in LibInterchain::_handleUniswapV2()/_handleCurve()

Target	Category	IMPACT	LIKELIHOOD	STATUS
LibInterchain.sol	Coding Practices	Low	Low	Addressed

Events are crucial for blockchain transparency, reliability, and interoperability. They play a vital role in updating user interfaces, confirming transactions, and facilitating cross-contract communication in decentralized applications (DApps). Incorrect event statuses can mislead users and systems that depend on these events.

While examining the implementation of `_handleUniswapV2()` (which is used to interact with `UniswapV2` for token exchange), we observe that it emits an `ActionDone` event with the `success` flag set to true regardless of whether the token exchange operation succeeds or fails. This behavior is misleading and incorrect, as it fails to accurately reflect the actual outcome of the exchange operation.

Similarly, the `_handleCurve()` function exhibits the same issue.

LibInterchain::_handleUniswapV2()

```
144 try
145     IUniswapV2(action.dexAddress).swapExactTokensForTokens(
146         _amount,
147         action.amountOutMin,
148         action.path,
149         address(this),
150         action.deadline
151     )
152 returns (uint256[] memory) {
153     emit ActionDone(Interchain.ActionType.UNI_V2, action.dexAddress, true, "");
154     // Note: instead of using return amounts of swapExactTokensForTokens,
155     //       we get the diff balance of before and after. This prevents errors
        for tokens with transfer fees
156     uint toBalanceAfter = LibSwapper.getBalanceOf(toToken);
157     SafeERC20.forceApprove(IERC20(action.path[0]), action.dexAddress, 0);
158     return (true, toBalanceAfter - toBalanceBefore, toToken);
159 } catch {
160     emit ActionDone(Interchain.ActionType.UNI_V2, action.dexAddress, true, "
        Uniswap-V2 call failed");
161     SafeERC20.forceApprove(IERC20(action.path[0]), action.dexAddress, 0);
162     return (false, _amount, shouldDeposit ? weth : _token);
163 }
```

Remediation Improve the `ActionDone` event in the `_handleUniswapV2()/_handleCurve()` functions.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI