SALUS SECURITY

OCT 2025

CODE SECURITY ASSESSMENT

PERLIST

Overview

Project Summary

- Name: PreList - Router
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/dsoftgames/RovexRouter
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	PreList - Router
Version	v4
Type	Solidity
Dates	Oct 07 2025
Logs	Sep 29 2025; Sep 30 2025; Oct 01 2025; Oct 07 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	9
Total informational issues	11
Total	20

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Implementation contracts lack of _disableInitializers()	Low	Configuration	Resolved
2	Missing events for functions that change critical state	Low	Logging	Resolved
3	Unremoved orders may cause the query function to experience dos	Low	Business logic	Resolved
4	Use call instead of transfer for native tokens transfer	Low	Business logic	Resolved
5	Unbounded active orders allow front-end dos	Low	Data Validation	Resolved
6	Partial fill rounding creates payment dust that cannot be withdrawn	Low	Business logic	Resolved
7	Lack of a minimal check may cause sellers to lose funds	Low	Business logic	Resolved
8	The exchangepairs and pairindex mapping will restrict the addition of trading pairs	Low	Logging	Acknowledged
9	Centralization risk	Low	Centralization	Mitigated
10	The contract logic does not correspond to the documentation description	Informational	Code Quality	Acknowledged
11	Missing __UUPSUpgradeable_init in initialize function	Informational	Business logic	Resolved
12	Incomplete docstrings	Informational	Code Quality	Resolved
13	Lack of security contact Information for responsible disclosure	Informational	Configuration	Resolved
14	Non-explicit Imports reduce code readability	Informational	Code Quality	Resolved
15	Mapping declaration lacks named parameters for improved readability	Informational	Code Quality	Resolved
16	Missing two-step transfer ownership pattern	Informational	Business logic	Resolved
17	Lack of __gap variables	Informational	Configuration	Resolved
18	Lack of license specification	Informational	Configuration	Resolved

19	Gas Optimization	Informational	Gas Optimization	Resolved
20	Use of floating pragma	Informational	Configuration	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Implementation contracts lack of _disableInitializers()

Severity: Low	Category: Configuration
Target: - contracts/RovexRouter.sol - contracts/RovexExchange.sol	

Description

The `RovexRouter` and `RovexExchange` contracts do not call OpenZeppelin's `_disableInitializers()` in its constructor.

As a result, anyone can send a direct transaction to the implementation address (not the proxy) and invoke `initialize()`. Gaining owner permissions within the implementation's own storage context on all contracts.

Recommendation

Add a constructor that locks the implementation:

```
constructor() {
    _disableInitializers();
}
```

Status

The team has resolved this issue in commit 7beb0f9.

2. Missing events for functions that change critical state

Severity: Low	Category: Logging
Target: - contracts/RovexRouter.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in protocol users.

In the `RovexRouter` contract,an event is lacking in the privileged setter functions (e.g. `setPriceIncrRate()`).

Recommendation

It is recommended to emit events for critical state changes.

Status

The team has resolved this issue in commit 1aea013.

3. Unremoved orders may cause the query function to experience dos

Severity: Low	Category: Business logic
Target: - contracts/RovexRouter.sol	

Description

Within the `RovexRouter` contract, the `fillBuyOrder()` function fails to remove orders from the `userBuyOrders` mapping after execution. As the project operates over time and users accumulate more pending orders, the `getUserBuyOrders()` function may become unable to correctly traverse all user orders. This could potentially lead to a denial-of-service (DoS) condition.

Recommendation

It is recommended that orders be removed from the mapping once the order has been completed.

Status

The team has resolved this issue in commit a4e291f.

4. Use call instead of transfer for native tokens transfer

Severity: Low	Category: Business logic
Target: - contracts/Lock.sol	

Description

The `transfer()` function is not recommended for sending native tokens due to its 2300 gas unit limit which may not work with smart contract wallets or multi-sig. Instead, `call()` can be used to circumvent the gas limit.

contracts/Lock.sol:L23-L33

```
function withdraw() public {
    // Uncomment this line, and the import of "hardhat/console.sol", to print a log in
your terminal
    // console.log("Unlock time is %o and block timestamp is %o", unlockTime,
block.timestamp);

    require(block.timestamp >= unlockTime, "You can't withdraw yet");
    require(msg.sender == owner, "You aren't the owner");

    emit Withdrawal(address(this).balance, block.timestamp);

    owner.transfer(address(this).balance);
}
```

Recommendation

Consider using `call()` instead of `transfer()` for sending native token.

Status

The team has resolved this issue in commit a4e291f.

5. Unbounded active orders allow front-end dos

Severity: Low	Category: Data Validation
Target: - contracts/RovexRouter.sol	

Description

contracts/RovexRouter.sol:L434-L449

```
function getUserBuyOrders(address user, uint256 index, uint256 size) external view
returns (BuyOrder[] memory) {
    uint256 length = userBuyOrders[user].length();
    if (index >= length) {
        return new BuyOrder[](0);
    }

    // Calculate the actual size to return
    uint256 actualSize = (index + size > length) ? length - index : size;

    // Create an array to hold the results
    BuyOrder[] memory orders = new BuyOrder[](actualSize);
    for (uint256 i = 0; i < actualSize; i++) {
        orders[i] = buyOrders[userBuyOrders[user].at(index + i)];
    }
    return orders;
}
```

Attackers can create a large number of buy orders with extremely low prices and minimal token amounts (e.g., 1 wei). These orders require negligible capital but bloat on-chain storage. When the front-end or back-end queries a user's orders (e.g., through getUserBuyOrders), large result sets may cause high gas for view calls, out-of-gas exceptions in RPC responses, or front-end crashes.

Recommendation

Set a per-user cap on active orders (e.g., MAX_ACTIVE_PER_USER) to deter spam.

Status

The team has resolved this issue in commit a4e291f.

6. Partial fill rounding creates payment dust that cannot be withdrawn

Severity: Low	Category: Business logic
Target: - contracts/RovexRouter.sol	

Description

Partial-fill rounding creates payment “dust” that cannot be withdrawn. The `fillBuyOrder()` function computes the buyer’s payment for each partial fill using round-down division:

```
function fillBuyOrder(
    uint256 orderId,
    uint256 sellAmount
) external nonReentrant whenNotPaused {
    ......
    // Calculate payment amounts before any state changes
    uint256 tokenDecimals = ERC20(order.token).decimals();
    uint256 grossPayment = sellAmount * order.price / (10 ** tokenDecimals);
    uint256 fee = grossPayment * sellOrderFeeRate / RATE_BASE;
    uint256 netPaymentToSeller = grossPayment - fee;
    ......
}
```

totalPayment is computed once, but each partial fill recomputes grossPayment with integer division (floor/round-down). Since ∑ floor(x_i) ≤ floor(∑ x_i), the sum of per-fill payments can be less than totalPayment, leaving dust trapped in the contract. When full-fills, the dust is not paid to the seller nor refunded to the buyer, it's locked in the contract.

Recommendation

Add a function to withdraw or send the dust in the final full-fill trade.

Status

The team has resolved this issue in commit [a4e291f](a4e291f).

7. Lack of a minimal check may cause sellers to lose funds

Severity: Low	Category: Business logic
Target: - contracts/RovexRouter.sol	

Description

Within the `RovexRouter` contract, the `fillBuyOrder()` function calculates `grossPayment` using integer division:

```
function fillBuyOrder(
    uint256 orderId,
    uint256 sellAmount
) external nonReentrant whenNotPaused {
    ......
    // Calculate payment amounts before any state changes
    uint256 tokenDecimals = ERC20(order.token).decimals();
    uint256 grossPayment = sellAmount * order.price / (10 ** tokenDecimals);
    uint256 fee = grossPayment * sellOrderFeeRate / RATE_BASE;
    uint256 netPaymentToSeller = grossPayment - fee;
    ......
}
```

If `sellAmount * price < 10**decimals`, then:

- grossPayment == 0
- fee == 0
- netPaymentToSeller == 0

Nevertheless, the contract will still transfer the sellAmount quantity of tokens from the seller to the buyer. Result: the seller gives away tokens for free.

This vulnerability is highly likely to be triggered when token precision is high (e.g. 18-digit), the quantity sold is extremely small, and/or the price is low.

Recommendation

It is recommended to add a check that `grossPayment` is greater than zero.

Status

The team has resolved this issue in commit a4e291f.

8. The exchangepairs and pairindex mapping will restrict the addition of trading pairs

Severity: Low	Category: Logging
Target: - contracts/RovexExchange.sol	

Description

The `exchangePairs` and `pairIndex` mappings use `sourceToken` as the key, which prevents the addition of further trading pairs for `sourceToken`. Attempting to add new trading pairs for `sourceToken` will overwrite the existing pair information.

For example, should subsequent requirements necessitate that TokenA can be exchanged simultaneously with both TokenB and TokenC within the contract, the current logic is unable to fulfil this demand.

Recommendation

It is recommended to use another reasonable value as the mapping key, such as the hash value of the two token addresses.

Status

This issue has been acknowledged by the team.

9. Centralization risk

Severity: Low	Category: Centralization
Target: - contracts/RovexRouter.sol - contracts/RovexExchange.sol	

Description

In `RovexRouter` and `RovexExchange` contracts, there exists one privileged role, `OWNER`. This role has the authority to execute some key functions such as `emergencyWithdraw()` and `withdraw()`.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

The team has mitigated this issue(in commit a4e291f).

2.3 Informational Findings

10. The contract logic does not correspond to the documentation description

Severity: Informational	Category: Code Quality
Target: - contracts/RovexRouter.sol	

Description

In the contract documentation, the functionality of the `createToken()` function is described as: 'Users can create custom ERC20 tokens.' However, within the contract, the `createToken()` function contains the `onlyOwner` modifier, restricting its invocation solely to the `owner`. This may mislead users regarding the contract's behaviour.

Recommendation

It is recommended that the functional documentation be amended in accordance with the contractual logic.

Status

This issue has been acknowledged by the team.

11. Missing __UUPSUpgradeable_init in initialize function

Severity: Informational	Category: Business logic
Target: - contracts/RovexExchange.sol	

Description

The contract inherits a total of four upgradeable contracts, but during initialization, only three of them were initialized. The `UUPSUpgradeable` contract was not initialized.

Recommendation

It is recommended to add logic to initialize the remaining contract.

Status

The team has resolved this issue in commit a4e291f.

12. Incomplete docstrings

Severity: Informational	Category: Code Quality
Target: - contracts/RovexExchange.sol - contracts/RovexRouter.sol	

Description

In the `RovexExchange` and `RovexRouter` contracts ,multiple instances of incomplete docstrings were identified across the codebase. Several functions/events do not document all of their parameters or return values. This reduces readability and maintainability of the code, and increases the effort required by developers, auditors, or integrators to fully understand the contract logic. When such functions are part of the contract's public API, incomplete documentation may further hinder external usage and integration.

Recommendation

It is recommended to provide complete documentation for all public functions and events, including details on their parameters and return values. Following the Ethereum Natural Specification Format (NatSpec) is highly encouraged to ensure consistency, readability, and usability. Comprehensive documentation improves long-term maintainability and facilitates both auditing and integration processes.

Status

The team has resolved this issue in commit a4e291f.

13. Lack of security contact information for responsible disclosure

Severity: Informational	Category: Configuration
Target: - All	

Description

All contracts do not specify a security contact point. Including a designated security contact (such as an email address or ENS name) in the contract's NatSpec header facilitates responsible vulnerability disclosure. This makes it easier for external researchers to quickly reach the appropriate team in the event a vulnerability is identified, helping minimize the time window between discovery and mitigation. The Ethereum community has begun standardizing this practice using the `@custom:security-contact` tag, adopted by tools such as OpenZeppelin Wizard and ethereum-lists.

Recommendation

Consider adding a NatSpec comment at the top of the contract with a `@custom:security-contact` field pointing to the preferred disclosure channel.

Status

The team has resolved this issue in commit a4e291f.

14. Non-explicit imports reduce code readability

Severity: Informational	Category: Code Quality

Target:
- contracts/RovexExchange.sol
- contracts/RovexRouter.sol
- contracts/Token.sol

Description

These contracts use wildcard or global-style import statements such as import "<path>";, which introduce all symbols from the imported file into the current compilation unit. While functional, this approach can reduce code readability and make it unclear which specific contracts, interfaces, or types are actually being used in the file. Explicit import { A, B } from "<path>"; declarations are generally preferred, as they make dependencies explicit and reduce the potential for namespace conflicts or unintentional symbol usage.

Recommendation

Consider refactoring a complete import statement to use named import syntax.

Status

The team has resolved this issue in commit a4e291f.

15. Mapping declaration lacks named parameters for improved readability

Severity: Informational	Category: Code Quality
Target: - contracts/RovexExchange.sol - contracts/RovexRouter.sol	

Description

Since Solidity version 0.8.18, the language allows developers to add named key and value identifiers in mapping declarations using the syntax:

```
mapping(KeyType keyName => ValueType valueName)
```

This enhances code readability and clarifies how a mapping is intended to be used. In the `RovexExchange` and `RovexRouter`, the mappings are declared without named parameters, making the purpose of the key and value less obvious upon inspection.
contracts/RovexExchange.sol:L25 - L26

```
mapping(address => ExchangeConfig) public exchangePairs;
mapping(address => uint256) private pairIndex;
```

contracts/RiceRoboticsNFTV2.sol:L36 - L45

```
mapping(address => TokenData) public tokens;

// Mapping from user address to their created tokens. userAddress => tokenAddresses[]
mapping(address => EnumerableSet.AddressSet) private userTokens;

// Buy orders mapping. orderId => BuyOrder
mapping(uint256 => BuyOrder) public buyOrders;

// Mapping from user address to their buy order IDs. userAddress => orderIds[]
mapping(address => EnumerableSet.UintSet) private userBuyOrders;
```

Recommendation

Consider updating the mapping declaration to explicitly name the key and value identifiers, enhancing readability and making the mapping's purpose clearer.

Status

The team has resolved this issue in commit a4e291f.

16. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - contracts/RovexExchange.sol - contracts/RovexRouter.sol	

Description

The `RovexExchange` and `RovexRouter` contracts inherit from the `OwnableUpgradeable` contract. These contracts do not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2StepUpgradeable contract from OpenZeppelin instead.

Status

The team has resolved this issue in commit a4e291f.

17. Lack of __gap variables

Severity: Informational	Category: Configuration
Target: - contracts/RovexRouter.sol	

Description

An upgradeable contract `RovexRouter` in the codebase was identified as missing the __gap storage variable. While the current implementation may not strictly require it (e.g., when the contracts are not inherited), the absence of __gap introduces potential risks in future upgrades if additional storage variables are added, as it could break storage layout compatibility.

Recommendation

It is recommended to include a __gap storage variable in all upgradeable contracts. This ensures forward compatibility of the storage layout, allowing new variables to be safely added in future upgrades.

Status

The team has resolved this issue in commit a4e291f.

18. Lack of license specification

Severity: Informational	Category: Configuration
Target: - contracts/Lock.sol	

Description

The `Lock` contract in the codebase specifies an incorrect license(e.g., // SPDX-License-Identifier: UNLICENSED).

Recommendation

It is recommended to explicitly specify a license (e.g., MIT or GPL-3.0) in all contracts and interfaces and maintain consistency across the codebase. This establishes clear legal usage terms, prevents misuse, and encourages collaboration.

Status

The team has resolved this issue in commit a4e291f.

19. Gas optimization

Severity: Informational	Category: Gas Optimization

Target:
- contracts/RovexExchange.sol
- contracts/RovexRouter.sol
- contracts/Lock.sol

Description

When dealing with the parameters of external functions, it is more gas-efficient to read their arguments directly from calldata instead of storing them to memory. calldata is a read-only region of memory that contains the arguments of incoming external function calls. This makes using calldata as the data location for such parameters cheaper and more efficient compared to memory. Thus, using calldata in such situations will generally save gas and improve the performance of a smart contract.

Throughout the codebase, multiple instances where function parameters should use calldata instead of memory were identified:

contracts/RovexRouter.sol: L88 - L99

```
function createToken(
    string memory name, // Token name
    string memory symbol, // Token symbol
    uint256 initialSupply, // Initial token supply
    uint256 price,  // BNB price in wei
    string memory icon, // Token icon URL
    string memory website, // Token website URL
    string memory telegram, // Token telegram group URL
    string memory twitter, // Token twitter URL
    uint256 holder, // Token holders count
    string memory description // Token description
) external payable whenNotPaused onlyOwner returns (address){
```

contracts/RovexRouter.sol: L147 - L156

```
function registerExistingToken(
    address tokenAddress,
    uint256 price,  // BNB price in wei
    string memory icon, // Token icon URL
    string memory website, // Token website URL
    string memory telegram, // Token telegram group URL
    string memory twitter, // Token twitter URL
    uint256 holder, // Token holders count
    string memory description // Token description
) external onlyOwner {
```

Throughout the codebase, these contracts use <if (condition) revert("error message") statements and require(condition, "error message") statements>. Since Solidity version 0.8.26, require statements support custom errors, which are more gas-efficient and improve code clarity. Initially, this feature was only available through the IR pipeline, but starting from Solidity 0.8.27, it is also supported in the legacy pipeline.

In several parts of the codebase, loops use the postfix increment operator (i++) for iteration. While functionally correct, this introduces an unnecessary step of storing the original value before incrementing, compared to using the prefix increment operator (++i). As the return value of the expression is ignored, this results in extra gas costs. In scenarios where loops run frequently, this overhead may accumulate to significant gas usage.

Recommendation

Consider optimizing gas usage and improving code readability by using calldata for external function parameters, replacing if-revert and require(condition, "error message") statements with require(condition, CustomError()), and preferring the prefix increment operator (++i) over the postfix increment operator (i++) in loop iterations.

Status

The team has resolved this issue in commit a4e291f.

20. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.28;
```

All contracts use a floating compiler version `^0.8.28`.

Using floating pragmas `^0.8.28` statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

The team has resolved this issue in commit a4e291f.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 4a29ad9:

File	SHA-1 hash
contracts/Lock.sol	c652e56a8db40f231099672eb99e1c51c64843cc
contracts/RovexExchange.sol	fcbc73fe465f592377e714f78fd68c267c3bd16f
contracts/RovexRouter.sol	db8701aedbcd50543b895b5e14752e3e2b8ed948
contracts/Token.sol	3112b1604a198260c328f4b4e997807d3d5035d6

And we audited the files that introduced new features in commit 71a5e99:

File	SHA-1 hash
contracts/RovexExchange.sol	f78a54632e5f8d22c1fc5657012367651d6876cd
contracts/RovexRouter.sol	00a7fbb2ab4ab8edc19ccb522ca78d7fc42f2be9
contracts/Token.sol	43d53189bc4a4794e5ba9d496267ae31d00f791b
contracts/RovexAirdrop.sol	393e78458f3c8277d17a8140add34b396248e9e3