SALUS SECURITY

MAY 2025

CODE SECURITY ASSESSMENT

LA EXCHANGE

Overview

Project Summary

- Name: La.Exchange
- Platform: EVM-compatible chains
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	La.Exchange
Version	v2
Type	Solidity
Dates	May 08 2025
Logs	May 06 2025; May 08 2025

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	1
Total Low-Severity issues	3
Total informational issues	1
Total	6

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	Error in the logical execution condition of the _invest() function	High	Business Logic	Resolved
2	Unexpected ETH should be refunded	Medium	Business Logic	Resolved
3	Deposit reverts when totalCurrencyLast is consumed to 0	Low	Business Logic	Acknowledged
4	Missing slippage protect	Low	Front-running	Acknowledged
5	Missing events for functions that change critical state	Low	Configuration	Acknowledged
6	Incorrect inheritance relationship	Informational	Business Logic	Resolved

2.2 Notable Findings

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

1. Error in the logical execution condition of the _invest() function

Severity: High	Category: Business Logic
Target: - src/LaunchPool.sol	

Description

There is a series of swap logic in the `_invest()` function that needs to be performed when `value` is zero. But in fact, according to the design of the contract, the call between `currency` and `underlying` should be performed when the value is not zero, while swap is not required when the value is zero, and using zero as a swap parameter will result in an error, which will cause the transaction to fail.

src/LaunchPool.sol:L134 - L150

```
function _invest(address account) internal {
    (uint quota, uint value, address[] memory path) = quotaDelta();
    if(value == 0) {
        if(value < totalCurrencyLast) {
            value = router.swapTokensForExactTokens(quota, totalCurrencyLast, path,
address(this), now)[0];
        } else {
            quota = router.swapExactTokensForTokens(value, 0, path, address(this),
now)[1];
        }
        totalCurrencyLast = totalCurrencyLast.sub(value);
        underlyingPerTokenLast =
underlyingPerTokenLast.add(quota.mul(denominator).div1(totalSupply));
    }
    lasttime = _align(now);
    if(account != address(0)) {
        underlyingLastOf[account] = underlyingOf(account);
        underlyingPerTokenOf[account] = underlyingPerTokenLast;
    }
}
```

Recommendation

It is recommended to refactor the conditional statements according to the contract design.

Status

This issue has been resolved by the team.

2. Unexpected ETH should be refunded

Severity: Medium	Category: Business Logic
Target: - src/LaunchPool.sol - src/LaunchPool2.sol	

Description

In both the `LaunchPool` and `LaunchPool2` contracts, the `deposit()` function uses the payable modifier to receive eth, which is used to facilitate the user's deposit when `currency` is `WETH`. However, when the `currency` is not `WETH`, the contract does not return the unexpected `ETH` to the user, which may result in a loss of funds.

Recommendation

It is recommended that the `ETH` of the user's current transaction be refunded in the logical branch of the user's deposited funds that is not `WETH`.

Status

This issue has been resolved by the team.

3. Deposit reverts when totalCurrencyLast is consumed to 0

Severity: High	Category: Business Logic
Target: - LaunchPool.sol	

Description

In the deposit() function, when `totalSupply != 0` and `totalCurrencyLast == 0`, the function exits early and refunds the user without processing the deposit. This scenario can arise after `_invest()` consumes all of `totalCurrencyLast` (e.g., via `totalCurrencyLast = totalCurrencyLast.sub(value))`, while `totalSupply` still remains non-zero. As a result, future deposits are blocked, leading to denial-of-service for depositors and a stuck contract state where no further capital can be added.

src/LaunchPool.sol:L68

```
function deposit(uint value) external payable invest(msg.sender) {
        uint amount;
        if(totalSupply == 0)
            amount = value;
        else if(totalCurrencyLast == 0) {        // Finished
            if(msg.value > 0)
        //...
```

Recommendation

Redesign this part of the code to avoid the situation where `totalCurrencyLast == 0` under normal circumstances, causing the contract to not operate normally.

Status

This issue has been acknowledged by the team. The `LaunchPool` contract has a defined launch time. Since deposits are allowed before the launch time but `limited tokens` are not released during this period, the issue mentioned above can be partially mitigated.

4. Missing slippage protect

Severity: Low	Category: Front-running
Target: - src/LaunchPool.sol - src/LaunchPool2.sol	

Description

Neither `LaunchPool` nor `LaunchPool2` imposes a minimum amount check on the received `quota` tokens.Due to the lack of slippage protection, trades may encounter excessive slippage and potential price manipulation, such as front-running.

src/LaunchPool.sol:L137 - L141

```
if(value < totalCurrencyLast) {
    value = router.swapTokensForExactTokens(quota, totalCurrencyLast, path,
address(this), now)[0];
} else {
    quota = router.swapExactTokensForTokens(value, 0, path, address(this), now)[1];
}
```

src/LaunchPool2.sol:L204 - L211

```
if(currency != currencies[0]) {
    ...
    quota = router.swapExactTokensForTokens(amount, 0, path0Und, address(this), now)[1];
} else
    quota = router.swapExactTokensForTokens(value, 0, path0Und, address(this), now)[1];
```

Recommendation

Consider setting some values for `amountOutMin` and `amountInMax`. This can be calculated from oracles. Please refrain from using spot price for calculating because spot prices can also be manipulated by the attacker.

Status

This issue has been acknowledged by the team.

5. Missing events for functions that change critical state

Severity: Low	Category: Logging
Target: - src/LaunchPool.sol - src/LaunchPool2.sol - src/LaunchSwapFactory.sol - src/LimitedERC20.sol - src/QuotaFarmingRouter.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 `LaunchPool`, `LaunchPool2`, `LaunchSwapFactory`, `LimitedERC20`, `QuotaFarmingRouter` contracts, events are lacking in the privileged setter functions.

Recommendation

It is recommended to emit events for critical state changes.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

6. Incorrect inheritance relationship

Severity: Informational	Category: Business Logic
Target: - src/LaunchPool2.sol	

Description

The `LaunchPool2` contract currently inherits from `ConstSepolia`, which is suitable for testing or testnet deployment. However, for mainnet deployment, it should inherit from `ConstEthereum` to ensure correct configuration and operational parameters.

src/LaunchPool2.sol:L29

```
contract LaunchPool2 is ConstSepolia
```

Recommendation

It is recommended to change the inheritance of `LaunchPool2` from `ConstSepolia` to `ConstEthereum` before deploying to mainnet.

Status

This issue has been resolved by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files :

File	SHA-1 hash
src/LaunchPool.sol	63f49a1561126871af9b1be4c9eee04f01871b9c
src/LaunchPool2.sol	d7b2a41d7fa0a9c6f474062200b503b866bd5c9b
src/LaunchSwapFactory.sol	266320026f67177a12c5eaa7ed1a1a9872242840
src/LaunchSwapPair.sol	0ca0ea94a83e1c4bbff5be24fd149592715c63fd
src/LaunchSwapRouter02.sol	081f1744d93d2f078b3cd55e7b1a47df7e7a5822
src/LimitedERC20.sol	6e65ae621f4e523abd7899af3634ca74adaef52f
src/QuotaFarmingRouter.sol	6ef691828cdc425e3bf60082798b576e4463732c
src/UniswapV2ERC20.sol	f0550560ad014ed78c6e945aaa17071019b23829