QuillAudits

Audit Report
July, 2024

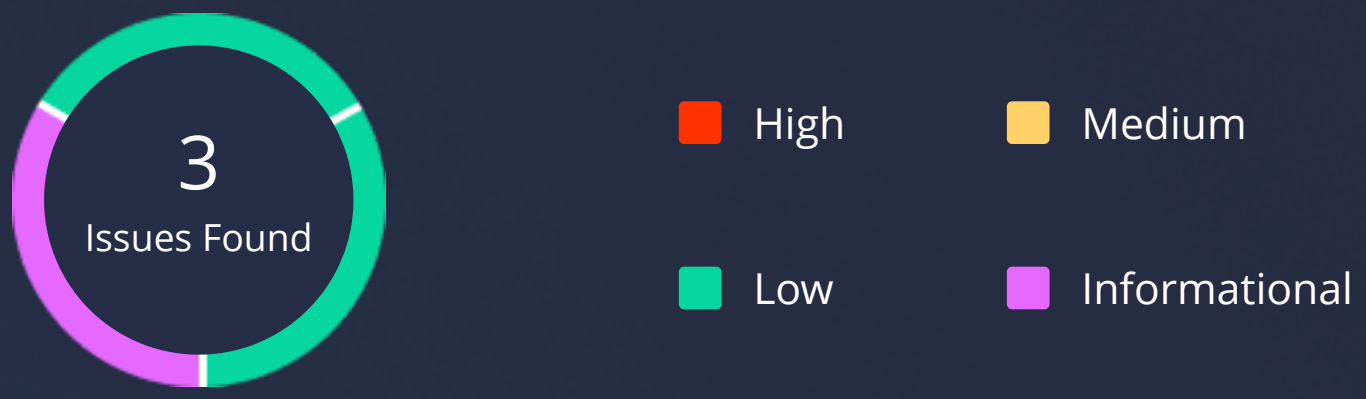
For

Table of Content

Executive Summary

Project Name	Raum Network
Overview	The smart contracts for RaumFi Core, encompassing its AMM, Factory, and Router functionalities, are crafted in Rust and utilize the Soroban SDK. Factory Contract - Factory Contract That includes the functionality of related to the pair address and pair contract. Route Contract - Router Contract is the main contract from which all the pair related functions like add , remove and swap on the tokens will be implemented.
Timeline	8th July 2024 - 24th July 2024
Updated Code Received	22nd July 2024
Second Review	22nd July 2024 - 24th July 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyze the raum-network for quality, security, and correctness.
Commit Hash	6ae6dd6ffc4ed4c748e13b12d80ad737e857d317
Source Code	https://github.com/Raum-Network/stellar-v2-core
Fixed In	feb59157327c28264e5a6107233956b9f0d7c400
Blockchain	Stellar

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	1	1

Checked Vulnerabilities

- Divide before multiply
- Unsafe unwrap
- Unsafe expect
- Overflow check
- Insufficiently random values
- Avoid panic error
- Avoid unsafe block
- DoS unbounded operation
- Soroban version
- Unused return enum
- Iterators over indexing
- Assert violation
- Unprotected mapping operation
- DoS unexpected revert with vector
- Unrestricted Transfer From
- Incorrect Exponentiation

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Code base were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Low Severity Issues

A.1 Inconsistent fee calculation

Line Function - math.rs

6

```
fn checked_ceiling_div(self, divisor: i128) -> Option<i128> {
    let result = self.checked_div(divisor)?;
    if self % divisor != 0 {
        result.checked_add(1)
    } else {
        Some(result)
    }
}
```

Description

Based on the calculations and the original implementation of the fork, the fee calculation should deduct only 0.3% of fees based on the swapped amount. However, the current implementation in the checked_ceiling_div function appears to be inconsistent with this expectation.
Suppose the amount_0 is 1001, the fee calculation according to function checked_ceiling_div is going to be 4, which is not correct as expected value should be 3.

Remediation

Consider fixing the function checked_ceiling_div to ensure consistent computation of the fees.

Status

Acknowledged

A.2 Possibility of overflow and underflow when performing arithmetic operations

Line **Function** - balance.rs

28-40

```
  pub fn receive_balance(e: &Env, addr: Address, amount: i128) {
   ....
   write_balance(e, addr, balance + amount);
}

pub fn spend_balance(e: &Env, addr: Address, amount: i128) {
  ....
   write_balance(e, addr, balance - amount);
}
```

Line **Function - allowance.rs**

51

```
pub fn spend_allowance(e: &Env, from: Address, spender: Address, amount: i128)
{
 .....
      write_allowance(
         e,
         from,
         spender,
         allowance.amount - amount,
         allowance.expiration_ledger,
      );}
```

Line **Function - storage.rs**

126

```
pub fn add_pair_to_all_pairs(e: &Env, pair_address: &Address) {
   let mut total_pairs = get_total_pairs(e);
   let key = DataKey::PairAddressesNIndexed(total_pairs);
   e.storage().persistent().set(&key, pair_address);
   e.storage()
      .persistent()
      .extend_ttl(&key, PERSISTENT_LIFETIME_THRESHOLD,
PERSISTENT_BUMP_AMOUNT);
   total_pairs += 1;
   put_total_pairs(e, total_pairs);
}
```

Description

In the current codebase, using native arithmetic operations raises the potential for overflow and underflow issues. To mitigate these risks, consider implementing checked safe methods for arithmetic operations. This oversight could lead to unpredictable behavior or errors if the counter exceeds its maximum allowable value. It is crucial to implement safeguards to ensure that the counter does not overflow.

Remediation

Consider using the checked_add/checked_sub method instead of the native arithmetic addition operation.

Status

Resolved

Informational Issues

A.3 No coverage report in the test-suite

Description

The test suite exhibits a failure of executing tests,however it’s not due to poorly written test but due to complicated environment requirements. Additionally, the absence of a coverage report implies a lack of information regarding test coverage.

Remediation

To rectify this issue, the recommended steps involve to dockerize the test suite and integrating a library such as cargo-llvm-cov or tarpaulin to generate a comprehensive coverage report.

Status

Acknowledged

Automated Tests

Dylint:

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity

Closing Summary

Nothing Critical has been found in the audit, code quality is good. Only Low and Informational Issue Found,which Raumn Network team Resolved and Acknowledged.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Raum Network. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Raum Network. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Raum Network to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

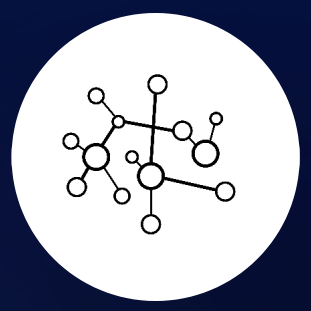
$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2024

For



QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com