Nostra Pools Security Review

Conducted by: Erim V.

Jan. 2024

Commit Hash: e472baa

Disclaimer

This smart contract audit report is provided for informational purposes only and does not constitute investment advice, financial guidance, or any other type of endorsement. The audit aims to identify potential vulnerabilities and issues within the smart contract code as of the date of the audit. However, it is important to understand that this audit does not guarantee the security or functionality of the smart contract and should not be seen as a comprehensive assurance of the code's integrity. Erim V. accepts no liability for any losses or damages arising from the use of this report or the smart contracts in question. Users of the smart contract are solely responsible for conducting their own due diligence and assuming all risks associated with its use.

Introduction

This document presents a detailed analysis and review of the smart contracts in question, aimed at identifying any potential security vulnerabilities, operational inefficiencies, and compliance issues. This audit was conducted just by Erim V. (0xErim).

Smart contracts are immutable once deployed on the blockchain, making it crucial to ensure their robustness and security prior to deployment. Our audit process involved a thorough examination of the contract's codebase, design patterns, and its interaction with both the blockchain and external entities. We focused on verifying the contract's adherence to specified requirements, its resilience against common attack vectors, and its overall performance efficiency.

Protocol Review

Nostra Pools smart contracts are used to swap two tokens with possible different routes. It acts like actual DEX, and implementation is similar to Uniswap V2. A new feature compared to Uniswap is the unique fee system depending on pairs. Every pair can have a different fee than others. There are 3 smart contracts (Router, Pair, and Factory). Where the factory deploys pairs permissionless, and the router routes the pair transactions. All smart contracts are upgradeable by the owner of the contract.

Risk Classification

Severity	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Likelihood

High: The attack vector is possible with reasonable assumptions and mostly simple to achieve on-chain conditions, cost of the attack is relatively low compared to the amount of loss done.
Medium: The attack vector needs specific conditions, but still relatively likely.
Low: There are many dependant assumptions that require significant conditions.

Impact

High: Causes significant loss of funds, losing ownership of contract, and changes user data without permission.
Medium: A small amount of funds can be lost, or the core functionality of the protocol is affected.
Low: Leads any kind of unexpected behavior with some of the protocol functionalities that are not critical.

Code improvements like gas optimizations and unnecessary codes are reported as **Best Practices** or **Informational**.

Executive Summary

This report will present findings about the protocol **Nostra Pools**. There will be a detailed representation of **9** findings in total. 2 Low, 3 Informational, and 4 Best Practices findings as grouped according to risk classification. In the final version of the report, there is also a status update for each fix in the below of each finding.

Findings

ID	Title	Severity	Status
L-01	Token symbols can cause a revert on pair creation	**Low**	Acknowledged
L-02	The swap fee can be higher than the limit	**Low**	Fixed
I-01	Selector callback *balanceOf* is unnecessary	**Informational**	Fixed
I-02	Selector callback for *transferFrom* is unnecessary	**Informational**	Fixed
I-03	Wrong starknet version is specified in *scarb.toml*	**Informational**	Fixed
BP-01	Use internal traits for internal contract methods	**Best Practices**	Fixed
BP-02	Unnecessary storage writings	**Best Practices**	Fixed
BP-03	Unnecessary assert statement	**Best Practices**	Fixed
BP-04	Store strings as byte31 array	**Best Practices**	Acknowledged

L-01 Token symbols can cause a revert on pair creation

File(s): utils.cairo, pair.cairo

Description: Generating pair symbols doesn't check for the length of token symbols. It causes reversion while creating a pair.

```
let (pair_mix, mix_multiplier) = join_short_strings(token_0_symbol, '/',
token_1_symbol);
```

Recommendation: Consider using byte31 arrays for storing strings.

L-02 The swap fee can be higher than the limit

File(s): factory.cairo, pair.cairo

Description: *swap_fee* parameter passed directly into constructor of pair contract, however it can be higher than 10000 and causes swap errors in pair contract.

```
        fn create_pair(
            ref self: ContractState,
            token_a: ContractAddress,
            token_b: ContractAddress,
            swap_fee: u16
        ) -> ContractAddress {
            // ...
            let constructor_calldata = array![
                token_0.into(), token_1.into(), swap_fee.into(),
self.ownable.owner().into()
            ];
            // ...
        }
```

Recommendation: Consider checking *swap_fee* before deploying the pair.

I-01 Selector callback balanceOf is unnecessary

File(s): utils.cairo

Description: In the current starknet version, failed transactions always revert the whole transaction. So, there is no way to handle reverted external calls.

```
fn balance_of_token(token: ContractAddress, account: ContractAddress) ->
u256 {
    let calldata = array![account.into()].span();
    let mut result = call_contract_syscall(token, SELECTOR_BALANCE_OF,
calldata); // @audit Unnecessary fallback

    if (result.is_err()) {
        result = call_contract_syscall(token, SELECTOR_BALANCEOF,
calldata);
    }

    (*result.unwrap_syscall().at(0)).into()
}
```

Recommendation: Consider using camel case function names.

I-02 Selector callback for transferFrom is unnecessary

File(s): utils.cairo

Description: In the current starknet version, failed transactions always revert the whole transaction. So, there is no way to handle reverted external calls.

```
fn transfer_token(
    token: ContractAddress, sender: ContractAddress, recipient:
ContractAddress, amount: u256
) {
    let mut calldata = array![sender.into(), recipient.into()];
    Serde::serialize(@amount, ref calldata);

    let mut result = call_contract_syscall(token, SELECTOR_TRANSFER_FROM,
calldata.span()); // @audit Unnecessary fallback

    if (result.is_err()) {
        result = call_contract_syscall(token, SELECTOR_TRANSFERFROM,
calldata.span());
    }

    result.unwrap_syscall();
}
```

Recommendation: Consider using camel case function names.

I-03 Wrong starknet version specified in *scarb.toml*

File(s): scarb.toml

Description: The starknet version in scarb config is wrong, and versions mismatch with other dependencies.

```
[dependencies]
starknet = "2.2.0"
```

Recommendation: Consider changing the starknet version in scarb.toml to 2.4.0.

BP-01 Use internal traits for internal contract methods

File(s): router.cairo, factory.cairo, pair.cairo

Description: Internal methods outside of the external trait have no access to the state. State parameter has to be passed all the time.

Recommendation: Consider implementing internal traits for internal methods.

BP-02 Unnecessary storage writings

File(s): factory.cairo, pair.cairo

Description: The initial values of these variables in storage are already zero and false. There is no need to write at constructor.

Pair.cairo

```
self._locked.write(false); // @audit Unnecessary storage write.
```

Factory.cairo

```
self._num_of_pairs.write(0); // @audit Unnecessary storage write.
```

Recommendation: Consider removing these lines.

BP-03 Unnecessary assert statement

File(s): factory.cairo

Description: The *sort_token_pair* function already checking whether tokens are zero or not.

```
assert(token_0.is_non_zero(), 'ZERO_ADDRESS'); // @audit Unnecessary
assertion. Zero address check is already done at sort_token_pair
```

Recommendation: Consider whether this check is still needed or not.

BP-04 Store strings as byte31 array

File(s): utils.cairo

Description: Cairo version 2.5.0 started to support byte31 arrays, which can be used to store strings.

Recommendation: Consider using byte31 array for string operations & storage.